ANL-AMD-TM-286

ARGONNE NATIONAL LABORATORY
Argonne, Illinois 60439

Applied Mathematics Division

NUMERICAL COMPARISON OF THREE NONLINEAR EQUATION SOLVERS*

by

Jorge J. Moré
Michel Y. Cosnard†

Technical Memorandum No. 286

February 1976

This report intended primarily for internal distribution.

*Work performed under the auspices of the U.S. Energy Research and Development Administration.

†Present address: Grenoble University, Grenoble, France.

DISCLAIMER

The facilities of Argonne National Laboratory are owned by the United States Government. Under the terms of a contract (W-31-109-Eng-38) between the U. S. Department of Energy, Argonne Universities Association and The University of Chicago, the University employs the staff and operates the Laboratory in accordance with policies and programs formulated, approved and reviewed by the Association.

MEMBERS OF ARGONNE UNIVERSITIES ASSOCIATION

The University of Arizona
Carnegie-Mellon University
Case Western Reserve University
The University of Chicago
University of Cincinnati
Illinois Institute of Technology
University of Illinois
Indiana University
Iowa State University
The University of Iowa
Kansas State University
The University of Kansas
Loyola University
Marquette University
Michigan State University
The University of Michigan
University of Minnesota
University of Missouri
Northwestern University
University of Notre Dame
The Ohio State University
Ohio University
The Pennsylvania State University
Purdue University
Saint Louis University
Southern Illinois University
The University of Texas at Austin
Washington University
Wayne State University
The University of Wisconsin

TABLE OF CONTENTS

THIS PAGE

WAS INTENTIONALLY

LEFT BLANK

NUMERICAL COMPARISON OF THREE NONLINEAR EQUATION SOLVERS

by

Jorge J. Moré and Michel Y. Cosnard

ABSTRACT

This paper is concerned with the numerical solution of n nonlinear equations in n unknowns by the methods of Newton, Brown, and Brent. The algorithms are described in detail and their implementations are compared on a set of test problems. It is found that a variation of Brent's method seems to perform best in a large number of cases, and a listing of an implementation of this method appears in an appendix.

1. INTRODUCTION

This paper is concerned with the numerical solution of n nonlinear equations in n unknowns. If R^n denotes real n-dimensional Euclidean space and $F:R^n \to R^n$ is a function with domain and range in R^n then this problem can be stated in vector form as $F(x) = 0$ or in component form as

$$f_i(x_1,\ldots,x_n) = 0 , \qquad 1 \le i \le n . \tag{1.1}$$

We restrict ourselves to three algorithms which only require the evaluation of F and which have second order convergence: Newton's method, Brown's method, and Brent's method.

The best known method for the solution of (1.1) is Newton's method. In 1966 Brown [2] proposed a new method which in a manner reminiscent of Gaussian elimination, reduced an appropriate Jacobian to lower triangular form. The method was somewhat surprising since it had second order convergence [3] but only used $(n^2+3n)/2$ component function evaluations per iteration. Although this was clearly an advantage over Newton's method and the numerical results were promising, the method did not have a clear algorithmic description and hence, was hard to implement.

In 1970 Brent [1] presented a similar method which reduced an approximate Jacobian to lower triangular form by a sequence of orthogonal transformations, again at a cost of only $(n^2+3n)/2$ component function evaluations per iteration and with second order convergence. Brent did give a clear algorithmic description of his method and suggested that Brown's method could be described similarly, but he did not derive his algorithm.

Then, in his Ph.D. dissertation, David Gay [6] took up Brent's suggestion and gave a clear algorithmic description of Brown's method. Perhaps more importantly, he showed that the original description of Brown's method led to $O(n^4)$ arithmetic operations per iteration and $n^2/2 + O(n)$ storage but that the revised version could be implemented in $O(n^3)$ arithmetic operations and $n^2/4 + O(n)$ storage. In addition Gay derived Brown's and Brent's method from a unified point of view and thus showed that these two methods were members of a class of methods which had second order convergence and only required $(n^2+3n)/2$ function evaluations per iteration.

In a recent paper, Gay [7] discussed the implementation of several variations of Brown's method and of one version of Brent's method. However, he only tested his implementations with starting vectors which were fairly close to the solution, and in this case the numerical performance of these two methods does not usually differ too much. Moreover, since the main purpose of Gay's paper was to test different variations of Brown's method, the resulting codes were mainly experimental.

The codes that we have implemented can only be considered local methods since they don't use any techniques which attempt to guarantee global convergence. It could be argued that our codes should have used some such technique, but this would have brought a host of other problems. For example, if we had insisted on a reduction of the sum of the squares of the residuals at each iteration, then this technique is dependent on the scaling of F, may slow down the iteration, and may cause convergence to a point at which the equations are not satisfied. Moreover, we also have to decide how to achieve the desired reduction. As they stand, our codes do not have any of the above problems, and the best of them seem to have a large region of convergence. However, if the iteration is not

progressing at a satisfactory rate then we attempt to diagnose the situation and terminate the iteration with an appropriate message. The techniques that we use for this are based in part on the ideas discussed by Shampine and Gordon [8].

In this paper we have also given a description of the algorithms of Brown and Brent which leads naturally to their implementation. This is done because it seems that the lack of popularity of these two algorithms is due to the fact that they have not been properly described.

There is a version of Brown's method which is available in the IMSL library, but this code is not suitable since it uses $O(n^4)$ arithmetic operations per iteration. The version of Brown's method which we have used in this paper only needs $O(n^3)$ arithmetic operations per iteration, and in the tests done by Cosnard [4], usually performs as well (and in some cases much better) as the IMSL version. There are other versions of Brown's method, but our code seems to perform just as well as the best of them -- the best version we have seen was written by H. A. Watts of Sandia Laboratories but this code also uses $O(n^4)$ arithmetic operations per iteration.

The testing done by Cosnard [4] also convinced us that Brent's method is more stable than Brown's method and therefore, that it would be profitable to write a code based on Brent's method for possible use in a subroutine library. This paper describes such a code, and the numerical results presented here show that it performs amazingly well in a large number of cases.

The outline of our paper is as follows. On Section 2 we describe Newton's method and some of its variations. This section motivates the introduction of Brown's and Brent's method and in it we point out that in some situations Newton-like methods are much more desirable than either Brown's or Brent's method. On the other hand, our numerical results indicate that a simple minded implementation of Newton's method is not nearly as stable as a corresponding implementation of Brown's or Brent's method.

Section 3 contains a derivation and description of Brent's method. The derivation presented here should make clear why Brent's method is able

to retain second order convergence while using almost half the number of function evaluations (per iteration) of Newton's method. Section 4 leans heavily on the material of Section 3 to give a brief derivation and description of Brown's method. In this section we also emphasize the differences between the algorithms of Brown and Brent. Section 5 contains a discussion of algorithmic details such as convergence criteria and techniques for detecting the divergence of the iteration. Finally, Section 6 presents the numerical results.

2. NEWTON'S METHOD

The methods proposed by Brown and Brent were motivated by a desire to improve Newton's method. Of course, there are many situations in which Newton's method is more desirable, and in this section we briefly describe some of the variations of Newton's method and point out when one of these variations is likely to be advantageous.

If $F'(x)$ denotes the Jacobian matrix (evaluated at x) of the mapping $F:R^n \to R^n$ then given an iterate x, Newton's method generates the next iterate x^+ by the following algorithm:

(2.1) (a) Solve the linear system $F'(x)\Delta x = -F(x)$ for the correction Δx.

(b) Set $x^+ = x + \Delta x$.

If the linear system in (2.1)(a) is solved by any of the standard direct methods then the computational requirement of one iteration of Newton's method is (assuming the evaluation of $F'(x)$ costs n^2 component function evaluations)

$n^2 + n$	component function evaluations
$O(n^3)$	arithmetic operations
$n^2 + O(n)$	storage

Since the evaluation of the Jacobian matrix is usually quite expensive, Newton's method is sometimes modified so that the Jacobian is only

evaluated at fixed intervals. This leads to the following algorithm:

(2.2) (a) Set $z_1 = x$

(b) For $\ell = 1,\ldots,m$

(b1) Solve the linear system $F'(x)\Delta z_\ell = -F(z_\ell)$ for the correction Δz_ℓ.

(b2) Set $z_{\ell+1} = z_\ell + \Delta z_\ell$.

(c) Set $x^+ = z_{m+1}$.

The motivation for this approach is based on the fact that if either the iterates or the Jacobian matrices are not changing too rapidly, then $F'(x)$ is a good approximation to $F'(z_\ell)$ and therefore (2.2) is "almost" Newton's method. These assumptions do not usually hold far away from the solution and then (2.2) with $m > 1$ can cause divergence in cases where the unmodified ($m = 1$) method was convergent.

Even though the above modification may reduce the number of Jacobian evaluations needed for convergence, the determination of the Jacobian matrix can still be a costly and error-prone task. Thus, in the above two algorithms $F'(x)$ is sometimes replaced by an approximation $A(x,h)$. If this approximation is determined by forward differences, then

$$A(x)e_i = [F(x+h_i e_i) - F(x)]/h_i, \qquad 1 \le i \le n , \tag{2.3}$$

for some parameter $h_i = h_i(x)$ and where e_i denotes the i^{th} column of the identity matrix. A good choice for this parameter seems to be

$$h_i = \epsilon \max\{|x_i|,1\} , \tag{2.4}$$

where $\epsilon = (\text{macheps})^{\frac{1}{2}}$ and macheps is the smallest floating point number for which $1 + \text{macheps} > 1$ in the precision being used.

Since in this paper we are interested in algorithms that only require the evaluation of F, we implemented a discretized Newton's method. In view of the above, one iteration of this algorithm consists of the following steps.

(2.5) (a) Construct $A(x)$ as defined by (2.3) and (2.4).

(b) Solve the linear system $A(x)\Delta x = -F(x)$ for the correction Δx.

(c) Set $x^+ = x + \Delta x$.

There are several points in which Newton's method compares favorably with either Brown's or Brent's method. One of the most important ones is that Newton's method is able to take into account the structure of the problem. Suppose, for example, that $F'(x)$ is tridiagonal. In this case the arithmetic overhead of Newton's method is $O(n)$ and it is possible to estimate $F'(x)$ with finite differences in four vector function evaluations (see the technique in [5] for general banded systems). Of course, the storage can also be reduced to $O(n)$. In sharp contrast with the above situation, if $F'(x)$ is tridiagonal, there is no reduction in overhead for either Brown's or Brent's method.

Another point which is sometimes important is that Brown's and Brent's method require a subroutine which, given a subscript k, will compute the k-th component $f_k(x)$ of $F(x)$. In some applications, computing $f_k(x)$ for some k is almost as expensive as computing $F(x)$ and in these cases Newton's method is much more attractive.

3. BRENT'S METHOD

Newton's method can be derived by a linearization argument: If $F:R^n \to R^n$ is linear, then

$$F(x^+) = F(x) + F'(x)(x^+-x) ,$$

and since we want to choose x^+ so that $F(x^+) = 0$, it is natural to require that

(3.1) $$0 = F(x) + F'(x)(x^+-x) .$$

Of course, $F'(x)$ can be replaced by an approximation, and in this case (3.1) can be written as

$$0 = f_j(x) + a_j^T(x^+-x) = 0 , \qquad 1 \le j \le n$$

where a_j is an approximation to $\nabla f_j(x)$. Brent's method is very closely related to this approach to Newton's method.

Given an iterate x, Brent's method generates the next iterate x^+ by the following algorithm:

(3.2) (a) Set $y_1 = x$.

(b) For k = 1,...,n let y_{k+1} be the solution of

$$f_j(y_j) + a_j^T(y-y_j) = 0 , \qquad 1 \le j \le k$$

which is closest to y_k.

(c) Set $x^+ = y_{n+1}$.

Following Gay [6] we will refer to x and x^+ as major iterates and the computation of x^+ from x as a major iteration. Similarly, the computation of y_{k+1} from y_k is a minor iteration and $y_1,\ldots,y_{n+1}$ are minor iterates.

In the above algorithm a_j is meant to be an approximation to $\nabla f_j(y_j)$. Of course, this description is not complete; to do this we must specify how to compute p_k where

(3.3) $$y_{k+1} = y_k + p_k ,$$

and how to compute the approximations a_j. For the moment we assume that $a_1,\ldots,a_k$ are given and show how to determine p_k.

The definition of y_{k+1} and equation (3.3) imply that

$$-a_j^T p_k = f_j(y_j) + a_j^T(y_k-y_j)$$

and thus, by the definition of y_k,

(3.4) $$a_j^T p_k = 0, \quad 1 \le j < k, \quad a_k^T p_k = -f_k(y_k) .$$

If $f_k(y_k) = 0$, then clearly $p_k = 0$ and thus $y_{k+1} = y_k$, so we assume that $f_k(y_k) \neq 0$. To find the vector of minimal length which satisfies (3.4) suppose for the moment that we have an orthogonal matrix Q_{k+1} such that

$$(3.5) \qquad \begin{bmatrix} a_1^T \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ a_k^T \end{bmatrix} Q_{k+1} = \begin{bmatrix} \sigma_1 & 0 & \cdots & & & & 0 \\ & \ddots & & & & & \cdot \\ x & & \ddots & & & & \cdot \\ \cdot & \cdot & & \ddots & & & \cdot \\ \cdot & & \cdot & & \ddots & & \cdot \\ \cdot & & & \cdot & & \ddots & \cdot \\ \cdot & & & & \cdot & & \cdot \\ x & \cdots & x & \sigma_k & 0 & \cdots & 0 \end{bmatrix}$$

and that $\sigma_i \neq 0$ for $1 \leq i \leq k$. Later on we will show how to generate Q_{k+1} efficiently. Since

$$a_j^T p_k = (a_j^T Q_{k+1})(Q_{k+1}^T p_k) ,$$

it follows from (3.5) that to satisfy $a_j^T p_k = 0$ for $1 \leq j < k$ we must have

$$(3.6) \qquad Q_{k+1}^T p_k = \sum_{j=k}^{n} \eta_j e_j$$

where e_j is the j-th column of the identity matrix. Also, $a_k^T p_k = -f_k(y_k)$ implies that

$$a_k^T p_k = \sigma_k \eta_k = -f_k(y_k) .$$

Thus $\eta_k = -f_k(y_k)/\sigma_k$. Since $\eta_{k+1},\ldots,\eta_n$ are otherwise unrestricted, and since

$$||p_k||_2^2 = \sum_{j=k}^{n} \eta_j^2 ,$$

to minimize $||p_k||_2$ we set $\eta_{k+1},\ldots,\eta_n$ to zero. Thus from (3.6),

$$p_k = \eta_k Q_{k+1} e_k = -\left(\frac{f_k(y_k)}{\sigma_k}\right) Q_{k+1} e_k .$$

It follows that

$$y_{k+1} = y_k - \left(\frac{f_k(y_k)}{\sigma_k}\right) Q_{k+1}\, e_k \ .$$

We have now shown how to carry out the iteration provided we have an orthogonal matrix Q_{k+1} which satisfies (3.5). To complete the description of the algorithm we indicate how Q_{k+1} can be computed efficiently.

We can assume that at the beginning of the k^{th} iteration we have an orthogonal matrix Q_k such that (3.5) holds with k replaced by (k-1). Of course, Q_1 is any orthogonal matrix. To obtain Q_{k+1} which satisfies (3.5), compute an orthogonal U_k of the form

$$U_k = \begin{bmatrix} I_{k-1} & \\ \hline & \hat{U}_k \end{bmatrix} \tag{3.7}$$

such that

$$(a_k^T Q_k) U_k = (x,\ldots,x,\sigma_k,0,\ldots,0) \ .$$

The matrix U_k can be either a single Householder matrix or the product of (n-k) Givens rotations. In either case it should be clear that $Q_{k+1} = Q_k U_k$ satisfies (3.5).

At the moment it seems that to carry out Brent's method it is necessary to compute $a_1,\ldots,a_n$ and that this requires n^2+n component function evaluations. To reduce this requirement we make two observations:

1. The matrix U_k only depends on the last (n-k+1) components of $\bar{a}_k \equiv Q_k^T a_k$.
2. The vector $\bar{a}_k = Q_k^T a_k$ is an approximation to $Q_k^T \nabla f_k(y_k)$.

In view of these two remarks, it follows that it is only necessary to compute the last (n-k+1) components of an approximation $\bar{a}_k$ to $Q_k^T \nabla f(y_k)$. This can be done by setting

$$\bar{a}_k = \frac{1}{h_k}\begin{pmatrix} 0 \\ \vdots \\ 0 \\ f_k(y_k+h_kQ_ke_k) - f_k(y_k) \\ \vdots \\ f_k(y_k+h_kQ_ke_n) - f_k(y_k) \end{pmatrix} \tag{3.8}$$

where the first (k-1) components of $\bar{a}_k$ have been arbitrarily set to zero. In analogy with (2.4) the parameter h is given by

$$h_k = \epsilon \max\{||x||_\infty, 1\} \tag{3.9}$$

where, as before ϵ is the square root of the machine precision.

The above discussion leads to the following description of a minor iteration in algorithm (3.2).

(3.10)

(a) Compute $\bar{a}_k$ by (3.8) and (3.9).

(b) Determine an orthogonal U_k of the form (3.7) such that $\bar{a}_k^T U_k = \sigma_k e_k^T$.

(c) Let $Q_{k+1} = Q_k U_k$ and set

$$y_{k+1} = y_k - \left(\frac{f_k(y_k)}{\sigma_k}\right) Q_{k+1}e_k .$$

At the end of a major iteration we have produced an orthogonal $Q^+ = Q_{n+1}$ and scalars $\sigma_1^+,\ldots,\sigma_n^+$ such that if $||y_k-x|| \leq \delta$ for k=1,...,n then

$$F'(x)Q^+ = L + E$$

where L is a lower triangular matrix with $\sigma_1^+,\ldots,\sigma_n^+$ on its diagonal and $||E|| = O(||h|| + \delta)$; if F is linear then E = 0.

In analogy with (2.2) it is possible to reuse the information in Q^+ and the σ^+'s; the computation of $z_{\ell+1}$ from z_ℓ in (b1) and (b2) of (2.2) would be replaced by the following algorithm.

(3.11) (a) Set $y_1 = z_\ell$

(b) For $k = 1,\ldots,n$ let

$$y_{k+1} = y_k - \left(\frac{f_k(y_k)}{\sigma_k^+}\right) Q^+ e_k$$

(c) Set $z_{\ell+1} = y_{n+1}$.

Thus, a combined algorithm in which information is being reused would consist of the following steps:

(3.12) (a) Set $y_1 = x$ and $Q_1 = I$.

(b) For $k = 1,\ldots,n$ use algorithm (3.10) to compute y_{k+1} from y_k.

(c) Set $z_1 = y_{n+1}$.

(d) For $\ell = 1,\ldots,m-1$, use algorithm (3.11) to compute $z_{\ell+1}$ from z_ℓ.

(e) Set $x^+ = z_m$.

We again warn the reader that the reuse of information may cause divergence; we will discuss this point further in Section 6.

It should be clear that one major iteration (algorithm (3.10) for $k = 1,\ldots,n$) of Brent's method requires $(n^2+3n)/2$ component function evaluations. In our implementation of Brent's method U_k is determined as a Householder matrix. The computation of the Q's then requires $n^3 + O(n^2)$ multiplications/divisions and the same number of additions/subtractions plus n square roots. In the computation of the $\bar{a}$'s it is possible to avoid the multiplication by h by letting Q_1 be h times an orthogonal matrix; thus the h factor is absorbed in the Q's. However, this leaves $n^3/2 + O(n^2)$ additions for the remaining of the computation. We summarize this discussion as follows: The implementation of Brent's method requires

$\frac{n^2 + 3n}{2}$ component function evaluations

$O(n^3)$ arithmetic operations

$n^2 + O(n)$ storage.

Brent [1] has observed that if $Q_1 = I$ then the storage can be reduced to $n^2/2 + O(n)$ by using Givens rotations instead of Householder rotations. This follows from the fact that in this case Q_k is of the form

$$Q_k = [\, Q_{k,1} \mid Q_{k,2} \,] \tag{3.13}$$

where $Q_{k,2}$ is an n by (n-k+1) lower Hessenberg matrix. If we use the three-multiply, three-add Givens rotation then the computation of the Q's requires the same amount of arithmetic as with Householder matrices but $n^2/2 + O(n)$ square roots. However, the computation of the $\bar{a}$'s now only requires $n^3/4 + O(n^2)$ additions/subtractions. It thus seems that the implementation with the Givens rotations requires less overhead. We point out that Gay [6] has an alternate implementation of Brent's method with the same storage and arithmetic requirements as the one based on Givens rotations but which uses Powell's orthogonalization procedure. In our implementation we opted for Householder matrices because of their simplicity and elegance.

There is one last point in connection with Brent's method that deserves attention. At the beginning of each major iteration there are at least two clear choices for the matrix Q_1; we can set $Q_1 = I$ or let Q_1 be the Q available from the end of the previous major iteration. The choice $Q_1 = I$ is necessary if we are to use Givens rotations, and has the advantage that it minimizes the departure from orthogonality of the Q's. Brent [1] favors the latter choice, but there does not seem to be any clear advantage to this choice and our numerical experiments bear this out. Thus, as indicated in (3.12), our implementation sets $Q_1 = I$.

4. BROWN'S METHOD

Brown's method is very similar to Brent's method, so in this section we essentially repeat the arguments of Section 3 but pointing out the differences between the two methods.

A major iteration of Brown's method consists of the following sequence of computations:

(4.1) (a) Set $y_1 = x$.

(b) For $k = 1,\ldots,n$, let y_{k+1} be the solution of

$$f_j(y_j) + a_j^T(y-y_j) = 0, \quad 1 \le j \le k$$

which "maximizes" the number of zero elements in $y_{k+1} - y_k$.

(c) Set $x^+ = y_{n+1}$.

In general it is very difficult to determine y_{k+1} so that $y_{k+1} - y_k$ has the maximum number of zero elements; however, we show that if we assume that no unlikely cancellations occur, then this is very easy to do.

We first obtain an expression for $p_k = y_{k+1}-y_k$. Just as in Brent's method, p_k satisfies

$$a_j^T p_k = 0, \quad 1 \le j < k, \quad a_k^T p_k = -f_k(y_k) . \tag{4.2}$$

Assume now that we have a nonsingular matrix R_{k+1} such that

$$\begin{bmatrix} a_1^T \\ \vdots \\ a_k^T \end{bmatrix} R_{k+1} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ x & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \\ x & & x & \sigma_k \end{bmatrix} \tag{4.3}$$

and that $\sigma_i \neq 0$ for $1 \le i \le k$. Since

$$a_j^T p_k = (a_j^T R_{k+1})(R_{k+1}^{-1} p_k)$$

it follows, as in Section 3, that to satisfy (4.2) we must have

$$R_{k+1}^{-1} p_k = \sum_{j=k}^{n} \eta_j e_j, \quad \eta_k = -\frac{f_k(y_k)}{\sigma_k} . \tag{4.4}$$

Components $\eta_{k+1},\ldots,\eta_n$ are otherwise unrestricted, so we now show that

if R_{k+1} is suitably chosen then setting these components to zero "maximizes" the number of zero elements in p_k.

The construction of R_{k+1} assumes the existence of R_k such that (4.3) holds with k replaced by (k-1). Of course, R_1 is an arbitrary non-singular matrix. Now determine an elementary permutation matrix P_k and an elementary upper triangular matrix T_k such that

$$a_k^T P_k T_k = (x,\ldots,x,\sigma_k,0,\ldots,0)$$

where σ_k is the maximal element (in absolute value) of a_k. For future reference recall that T_k is of the form

$$T_k = I - e_k w_k^T \tag{4.5}$$

$$w_k = (0,\ldots,0,\omega_{k+1}^{(k)},\ldots,\omega_n^{(k)}) .$$

It follows that if $R_{k+1} = R_k P_k T_k$ then R_{k+1} satisfies (4.3). We now show that if the rows of R_{k+1} are suitably permuted then R_{k+1} is an upper triangular matrix of special form.

Lemma. If $R_1 = I$ then

$$P_k \cdot \ldots \cdot P_1 R_{k+1} = \left[\begin{array}{c|c} L_k & S_k \\ \hline 0 & I_{n-k} \end{array}\right] \tag{4.6}$$

where L_k is a unit upper triangular matrix of order k and I_{n-k} is the identity matrix of order n-k.

Proof. The result holds for k = 1 since, by assumption, $R_1 = I$. Assume by induction that

$$P_{k-1} \cdot \ldots \cdot P_1 R_k = \left[\begin{array}{c|c} L_{k-1} & S_{k-1} \\ \hline 0 & I_{n-k+1} \end{array}\right] .$$

Then

$$P_k \cdot \ldots \cdot P_1 R_k P_k = \left[\begin{array}{c|c} L_{k-1} & \hat{S}_{k-1} \\ \hline & \\ & I_{n-k+1} \end{array}\right]$$

where $\hat{S}_{k-1}$ is S_{k-1} with two columns interchanged. Now (4.5) implies that

$$P_k \cdot \ldots \cdot P_1 R_{k+1} = P_k \ldots P_1 R_k P_k - v_k w_k^T \tag{4.7}$$

where

$$v_k = P_k \ldots P_1 R_k P_k e_k \ .$$

The result now follows since v_k has zeroes in the last (n-k) positions and w_k has zeroes in its first k positions.

With the Lemma it is easy to show that setting $\eta_{k+1}, \ldots, \eta_n$ to zero in (4.4) "maximizes" the number of zero elements in p_k. In fact, from (4.4) and the Lemma,

$$(P_k \ldots P_1) p_k = (x, \ldots, x, \eta_{k+1}, \ldots, \eta_n)^T \ .$$

Thus setting $\eta_{k+1}, \ldots, \eta_n$ to zero "maximizes" the number of zero elements in p_k.

It should be clear, in analogy with Brent's method, that we need not compute a_k, but instead we compute

$$\bar{a}_k = \frac{1}{h_k} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ f_k(y_k + h_k R_k e_k) - f_k(y_k) \\ \vdots \\ f_k(y_k + h_k R_k e_n) - f_k(y_k) \end{pmatrix} \tag{4.8}$$

In our implementation of Brown's method the parameter h in (4.8) is also given by (3.9). An alternate strategy would first normalize the vectors $R_k e_j$; that is compute

$$f_k\left(y_k + h \frac{R_k e_j}{||R_k e_j||}\right) - f_k(y_k)$$

where h is given by (3.9). This would require an additional $O(n^3)$ arithmetic operations and would not improve the algorithm unless the norms of the vectors $R_k e_j$ deviate drastically from unity.

The above discussion leads to the following summary of a minor iteration of Brown's method:

(4.9)

(a) Compute $\bar{a}_k$ by (4.8) and (3.9).

(b) Determine an elementary permutation P_k so that $\bar{a}_k^T P_k$ has its maximal element in absolute value in the kth position.

(c) Determine an elementary upper triangular matrix T_k of the form (4.5) such that $a_k^T P_k T_k = \sigma_k e_k^T$.

(d) Let $R_{k+1} = R_k P_k T_k$ and set

$$y_{k+1} = y_k - \left(\frac{f_k(y_k)}{\sigma_k}\right) R_{k+1} e_k.$$

Thus a major iteration of Brown's method consists of the following steps.

(4.10)

(a) Set $y_1 = x$ and $R_1 = I$.

(b) For k = 1,...,n, use algorithm (4.9) to compute y_{k+1} from y_k.

(c) Set $x^+ = y_{n+1}$.

At the end of a major iteration we have produced a matrix $R^+ = R_{n+1}$ and scalars $\sigma_1^+,\ldots,\sigma_n^+$ such that if $||y_k - x|| \leq \delta$ for k = 1,...,n then

$$F'(x)R^+ = L+E$$

where L is a lower triangular matrix with $\sigma_1^+,\ldots,\sigma_n^+$ on its diagonal and $||E|| = O(||h|| + \delta)$; if F is linear then $E = 0$. Of course, just as in Brent's method this information can be reused and the resulting algorithm would just be (3.11) with Q^+ replaced by R^+.

It is more difficult to implement Brown's method than Brent's method because of the particular structure of R_{k+1}. The Lemma of this section shows that it is only necessary to store the matrix S_k which appears in (4.5) and since this is a k by (n-k) matrix, the storage will not exceed $n^2/4$. In our implementation of Brown's method we have decided to store S_k by rows in a vector u(·). The information contained in $P_k\cdot\ldots\cdot P_1$ is stored in a vector mperm (·) by requiring that mperm (i) = j if and only if the (i,j) element of $P_k\cdot\ldots\cdot P_1$ is unity.

With the above information it is not difficult to carry out Brown's method. The reader should verify that the most expensive parts of a minor iteration are the calculation of S_k from S_{k-1} and the formation of the vectors $y_k + hR_k e_j$. The first task requires k(n-k) multiplications/divisions and the same number of additions/subtractions; the second task requires (since the multiplication by h can be avoided as in Brent's method) k(n-k) additions/subtractions. Thus one major iteration of Brown's method requires

$\frac{n^2+3n}{2}$ component function evaluations

$O(n^3)$ arithmetic operations

$\frac{n^2}{4} + O(n)$ storage

If we assume that multiplications and additions each costs one unit of time, then a detailed count shows that the overhead of Brown's method depends on $1/2\ n^3$ while that of Brent's method depends on $5/2\ n^3$. If the functions are easy to evaluate then this overhead may dominate and all other things being equal, Brown's method would be the most efficient method. However, this is not the case in our numerical examples; on problems which require the same number of function evaluations, the computing times for both methods did not differ by more than ten percent.

5. ALGORITHMIC CONSIDERATIONS

Before an algorithm for the solution of nonlinear equations is completely defined, it must decide on a course of action when faced with one or more of the following problems:

(a) Convergence criteria.

(b) Lack of satisfactory progress or divergence of the iteration.

(c) Requested accuracies unreasonably high.

(d) Singularity of the approximate Jacobian.

(e) Lack of a solution.

All of the above problem, and the algorithms of the three previous sections are affected by the scaling of F and/or x. A change of scale in F corresponds to considering the scaled mapping F defined by $\hat{F}(x) = \Sigma \cdot F(x)$ where Σ is a diagonal matrix with positive diagonal entries. It should be clear that such a change in scale leaves invariant the iterates in all of the three algorithms considered. If we consider a change of scale in x so that $\hat{F}(x) = F(\Sigma x)$ then the solution is changed from x^* to $\Sigma^{-1}x^*$. A highly desirable property for an iterative method would then be that if $\Sigma\hat{x}_0 = x_0$ then $\Sigma\hat{x}_k = x_k$ for $k \geq 1$. However, no nonlinear equation solver seems to have this property.

We now turn to a discussion of problems (a) through (e) mentioned at the beginning of the section.

(a) The convergence criteria depends on a measure of the residuals and a measure of the relative error between consecutive iterates. For Brown's and Brent's method define

$$\text{FNORM} = \max\{|f_k(y_k)| : 1 \leq k \leq n\} \tag{5.1}$$

while for Newton's method

$$\text{FNORM} = \max\{|f_k(x^+)| : 1 \leq k \leq n\} . \tag{5.2}$$

For either of the three methods we also define

$$\text{(5.3)} \qquad \text{DIFIT} = ||x^+ - x||_\infty, \quad \text{XNORM} = ||x^+||_\infty .$$

If the user provides two parameters, FTOL and XTOL then the subroutines will stop if either

$$\text{(5.4)} \qquad \text{FNORM} \leq \text{FTOL}$$

or

$$\text{(5.5)} \qquad \text{DIFIT} \leq \text{XTOL} \cdot \text{XNORM} .$$

The first criterion is dependent on the scaling of F and thus should be used with care. Criterion (5.5) usually guarantees that the components of x^* have $-\log_{10}(\text{XTOL})$ significant digits as approximations to the corresponding components of x^*. Of course, (5.5) cannot guarantee any agreement for the components which are much smaller than XNORM, but the fast convergence of the iteration does force quite close agreement. Note that there are at least two cases in which (5.5) may fail. It will probably fail if the solution is at the origin, but in this case the iteration will stop because the requested accuracy is too high; see the discussion under (c). Criteria (5.5) can be satisfied at a point far away from the solution if at that point the derivative is large relative to the residuals. This problem can usually be avoided by requiring that both FNORM and DIFIT have decreased from the previous iteration, and this is done on the implementations.

(b) To measure the progress of the iteration we monitor FNORM and DIFIT as defined by (5.1), (5.2), and (5.3). If FNORM and DIFIT increase during three consecutive iterations then we terminate the iteration since this indicates that the iteration is diverging. However, the iteration is almost certainly not making good progress if at least one of FNORM and DIFIT increases, so if this happens during five consecutive iterations, then we terminate the iteration for lack of good progress. The user

because of a diagnosed divergence then this will probably lead to overflow problems. If it was terminated for lack of good progress then restarting the iteration may lead to convergence, but this has not happened too often; a better strategy is to choose another starting point.

(c) We only attempt to detect unreasonably high requested accuracies in the well-scaled case. In this situation we can expect that at best $||F(x)|| \sim$ macheps or $||x^+ - x|| \sim$ macheps $\cdot ||x||$. Thus we test whether

$$\text{FNORM} \leq \epsilon \quad \text{or} \quad \text{DIFIT} \leq \epsilon \cdot \max\{\text{XNORM}, 1\} , \tag{5.6}$$

where $\epsilon = (\text{macheps})^{\frac{1}{2}}$, and if either of these inequalities holds on four consecutive iterations then we exit with an indication that the requested accuracies are too stringent. Note that if the convergence is slow then even reasonable values for FTOL and XTOL may be diagnosed as being too stringent. Thus test (5.6) will also detect slow convergence (which is usually due to a Jacobian singular near the iterates or to bad scaling). Also note that the iteration may stop with an indication of stringent accuracies just because XNORM $\leq \epsilon/2$ on five consecutive iterations. Thus we have imposed an absolute convergence criteria on the user. However, because the convergence is usually very fast, it is highly unlikely that this will happen unless the solution is at the origin.

(d) Consider first singularity of the approximate Jacobian in Newton's method. If we are solving the linear system in (2.1) by Gaussian elimination, then singularity is detected by a zero pivot. Attempting to detect singularity by a pivot ρ_k which is small relative to the size of the matrix and the precision of the computation leads to difficulties when F is badly scaled. If $\rho_k = 0$ then we replace ρ_k by macheps $\cdot$ $\max\{||A||, 1\}$.

In Brown's or Brent's method singularity of the approximate Jacobian is detected when $\sigma_k = 0$ for some k. If $\sigma_k = 0$ then we set $y_{k+1} = y_k$ and continue the iteration. If $\sigma_k = 0$ for $k = 1,\ldots,n$, then we exit with an indication that the approximate Jacobian is singular.

(e) The lack of a solution in the problem will almost invariably lead to problem (b), but it is always possible for the iteration to oscillate about a point at which the Jacobian is singular and the equations are not satisfied. However, this did not happen in our tests even with a highly artificial example such as $f(x) = x^2+1$.

6. NUMERICAL RESULTS

We now present numerical results for four subroutines. The first three, NEWTON, BROWN, and BRENT are just implementations of algorithms (2.5), (3.12) with m = 1, and (4.10), respectively. The fourth one, BRENTM, is a modification of BRENT in which the Jacobian is reused on some iterations; that is, algorithm (3.12) for some m > 1. Before BRENTM is completely defined we have to decide when to reuse the Jacobian and for how long.

The idea of reusing the Jacobian is only worthwhile in the latter part of the iteration; if it is used in the early stages of the iteration the performance of the original algorithm invariably deteriorates. Thus the Jacobian is only reused to refine the solution in a manner very similar to iterative refinement.

After some experimentation we decided to execute the iterative refinement (step (d) of algorithm (3.12)) if the following two conditions hold.

(6.1) $$||y_{n+1}-x|| \leq 0.05\ ||y_{n+1}|| \ .$$

(6.2) Both FNORM and DIFIT as defined in Section 5 have decreased from the previous iteration.

Condition (6.1) indicates that the iterates are beginning to converge while (6.2) guarantees that (6.1) will not hold accidentally.

Next we had to decide on the choice of m that would be used in the iterative refinement. Theoretical results show that the optimal choice of m would maximize the efficiency, which in the case of (3.12) Brent [1]

showed that it was given by

$$E(m) = \frac{2 \ln (m+1)}{(n+2m+1)} .$$

Numerically this turns out to be a good choice and thus we decided to set $m = m^*$ where m^* maximizes $E(m)$ for $m = 1,2,\ldots,n$.

To present the numerical results we have selected five test problems; other test problems will be discussed in Section 7. These examples were selected because they represent typical situations that we have encountered in our testing. The following specifications apply to these test results.

(a) All problems were run on the IBM 370/195 of Argonne National Laboratory in double precision (14 hexadecimal digits) and under FORTRAN H (opt=2) compiler.

(b) The tolerances were set at FTOL = 10^{-10} and XTOL = 10^{-10}. Since, with the exception of the fifth problem, the convergence is of second order, our results are only marginally affected by the choice of FTOL and XTOL.

(c) Each problem was run with three starting vectors. We always give the starting vector x_0 which is closest to the solution; the other two points are $10x_0$ and $100x_0$.

(d) For each run we report three numbers: the number of iterations and vector function evaluations required to produce the final iterate, and the maximum residual at the final iterate. Thus the entry 5,33,0.8(-14) means that for this problem the subroutine required 5 iterations and 33 vector function evaluations, and that the maximum residual at the final iterate was 0.8×10^{-14}.

1. Two-point boundary value problem

If we apply the standard $O(h^2)$ discretization to the two-point boundary value problem

$$u''(t) = \frac{1}{2}(u(t)+t+1)^3, \quad 0 < t < 1, \quad u(0) = u(1) = 0 ,$$

then the resulting system in the unknowns $x_k = u(t_k)$ is defined by

$$f_k(x) \equiv 2x_k - x_{k+1} - x_{k-1} + \frac{h^2}{2}(x_k+t_k+1)^3, \quad 1 \le k \le n , \tag{6.3}$$

where $x_0 = x_{n+1} = 0$, $t_k = kh$, and $h = 1/(n+1)$. The results of solving this problem with n = 10 are listed in Table 1 where x_0 refers to the point

$$x_0 = (\xi_i) \quad \text{with} \quad \xi_i = t_i(t_i-1), \quad 1 \le i \le n . \tag{6.4}$$

TABLE 1

	x_0	$10\ x_0$	$100\ x_0$
NEWTON	3, 34, 0.3(-15)	4, 45, 0.1(-16)	9, 100, 0.5(-14)
BROWN	4, 26, 0.4(-16)	5, 33, 0.3(-16)	10, 65, 0.4(-16)
BRENT	4, 26, 0.1(-15)	6, 39, 0.6(-16)	11, 72, 0.5(-16)
BRENTM	2, 16, 0.1(-15)	4, 28, 0.6(-15)	9, 61, 0.1(-15)

Remarks

(a) Equations (6.3) have a unique solution $x^* = (\xi_i^*)$ with $-0.5 \le \xi_i^* \le 0$ for $1 \le i \le n$.

(b) In each case NEWTON solved the problem in fewer iterations but with more function evaluations than either BRENT or BROWN. To explain this, note that FNORM as defined by (5.1) measures the size of the residuals at the beginning of the iteration since usually $|f_1(y_1)|$ is the largest residual. Thus both BRENT and BROWN always stop one iteration too late. There does not seem to be an elegant, scale-independent method to avoid this although the iterative refinement of BRENTM certainly helps.

(c) For this function BROWN converges in less function evaluations than BRENT; this is not the case for the other four functions.

(d) The evaluation of any f_k only requires a fixed number of arithmetic operations (about a dozen) and thus the computing time of an iteration of either BROWN, BRENT, or BRENTM is dominated by the overhead. To compare the actual overhead of BROWN and BRENT, we solved the above problem for n = 25, 50, and 75 for the x_0 given by (6.4). In each case BROWN and BRENT each required four iterations to reach an acceptable solution while three and two iterations were required by NEWTON and BRENTM, respectively. Table 2 presents the computing times (in seconds), and shows that in the case of BROWN and BRENT they never differ by more than 10%. The reason why the number of arithmetic operations does not reflect the computing time can be traced to the special architecture of the IBM 370/195; on other machines the relationship between these times may vary.

TABLE 2

	25	50	75
NEWTON	0.024	0.085	0.18
BROWN	0.124	0.543	1.63
BRENT	0.104	0.574	1.79
BRENTM	0.057	0.302	0.889

2. Nonlinear integral equation

The nonlinear integral equation

$$u(t) + \int_0^1 H(s,t)(u(s)+s+1)^3 ds = 0 ,$$

$$H(s,t) = \begin{cases} s(1-t), & s \leq t, \\ t(1-s), & s \geq t, \end{cases}$$

can be discretized by considering the equation at the points $t = t_k$, $k = 1,\ldots,n$, and then replacing the integral by an n-point rectangular rule based on the points $\{t_k\}$. The resulting system of equations in the unknowns $x_k = u(t_k)$ is defined by

$$f_k(x) \equiv x_k + \frac{h}{2}\left\{(1-t_k)\sum_{j=1}^{k} t_j(x_j+t_j+1)^3 + t_k\sum_{j=k+1}^{n}(1-t_j)(x_j+t_j+1)^3\right\} \tag{6.5}$$

where $x_0 = x_{n+1} = 0$, $t_j = jh$, and $h = 1/(n+1)$. The results of solving this problem with $n = 10$ are listed in Table 3 where as the initial guess x_0 we again took (6.4).

TABLE 3

	x_0	$10\ x_0$	$100\ x_0$
NEWTON	3, 34, 0.3(-14)	4, 45, 0.4(-16)	9, 100, 0.6(-13)
BROWN	4, 26, 0.5(-16)	5, 33, 0.4(-16)	4, 26, 0.1(33)
BRENT	4, 26, 0.4(-16)	5, 33, 0.6(-16)	4, 26, 0.5(17)
BRENTM	2, 15, 0.2(-15)	3, 22, 0.8(-16)	4, 26, 0.5(17)

Remarks

(a) The results under NEWTON in Tables 1 and 3 are identical. This is due to the fact that if $F^{(1)}$ and $F^{(2)}$ denote the functions defined by (6.3) and (6.5), respectively, then there is a nonsingular matrix A such that $F^{(1)}(x) = A\cdot F^{(2)}(x)$. In fact, if A is the tridiagonal matrix with 2's on the diagonal and -1's on the off-diagonal then it can be shown that $F^{(1)}(x) = Ax+G(x)$ where $g_k(x) = (h^2/2)(x_k+t_k+1)^3$ while $F^{(2)}(x) = x + A^{-1}G(x)$. Tables 1 and 3 reflect the fact that algorithm (2.5) is invariant under this type of transformation, but that this is not the case for neither (3.12) nor (4.10).

(b) It seems that if BROWN and BRENT both diverge then BROWN diverges at a faster rate; a particular instance of this happens with the third starting point.

(c) Functions (6.5) require n + O(1) arithmetic operations to evaluate and thus the computing time per iteration of all four subroutines is almost equally dependent on the overhead and on the function evaluations. Table 4 was produced in the same manner as Table 2 but for functions (6.5), and it turned out that the number of iterations required to produce an acceptable answer coincided with those for Table 2. However, now the computing times of NEWTON, BROWN, and BRENT are almost identical.

TABLE 4

	25	50	75
NEWTON	0.165	1.14	3.81
BROWN	0.194	1.29	3.97
BRENT	0.206	1.28	4.11
BRENTM	0.113	0.656	2.11

3. Brown's almost linear function

To define this function let

$$f_k(x) = x_k + \sum_{j=1}^{n} x_j - (n+1), \quad 1 \le k \le n-1 ,$$

(6.6)

$$f_n(x) = \left(\prod_{j=1}^{n} x_j \right) - 1 .$$

The results of solving this problem with n = 10 are listed in Table 5 where the initial guess x_0 is given by

(6.7) $$x_0 = (\xi_i) \quad \text{with} \quad \xi_i = 1/2, \quad 1 \le i \le n .$$

TABLE 5

	x_0	$10\ x_0$	$100\ x_0$
NEWTON	90, 991, 0.9(-11)	103, 1134, 0.1(-14)	91, 1002, 0.1(-13)
BROWN	8, 52, 0.1(-14)	18, 117, 0.1(-14)	19, 124, 0.3(27)
BRENT	5, 33, 0.8(-14)	6, 39, 0.1(-14)	22, 143, 0.1(-13)
BRENTM	3, 25, 0.1(-14)	3, 26, 0.8(-15)	20, 135, 0.1(-13)

Remarks

(a) It can be shown that all zeroes of (6.6) are of the form $(\alpha,\ldots,\alpha,\alpha^{1-n})$ where α satisfies $n\alpha^n - (n+1)\alpha + 1 = 0$. If n is even this equation has two real roots but if n is odd then it has three real roots. If $n = 10$ then $\alpha = 1$ and $\alpha = 0.9794\ldots$ are the two roots.

(b) Since the first n-1 equations are linear the iterates produced by the methods of Brown and Brent satisfy $f_j(x_k) = 0$ for $1 \leq j < n$ and all $k \geq 1$. This feature gives these two methods an advantage over more conventional nonlinear equation solvers. This advantage is wiped out if the nonlinear equation is placed first in the definition of (6.6). The effect of this interchange on the numerical results of Table 5 is given in Table 6 and shows that in this case NEWTON, BROWN and BRENT require almost the same number of iterations.

(c) The solution of Brown's function with Newton's method presents an interesting difficulty; it can be shown that x_k is of the form $(\beta,\ldots,\beta,\gamma)$ for $k \geq 1$ regardless of the choice of x_0, and therefore, unless β is close to unity, $F'(x_k)$ and the approximation $A(x_k)$ determined by (2.3) and (2.4) are both ill-conditioned and badly scaled. This difficulty is worsened by the following observation.

(d) Shampine and Gordon [8] have noted that if n is sufficiently large then it is difficult to estimate $\nabla f_n(x_0)$ by differences. More

generally, if $||x||_\infty \le r$ and $||v||_\infty \le \epsilon$ (recall ϵ is the square root of macheps), then

$$|f_n(x+v) - f_n(x)| \le n\epsilon||\nabla f_n(x+\theta v)|| \le n\epsilon(r+\epsilon)^{n-1},$$

$$|f_n(x) + 1| \le r^n .$$

In particular, if $r \le 1/2$ and $n \ge 30$, then $f_n(x+v) = f_n(x)$ for the IBM 360-370; for smaller values of n the approximation $A(x_k)$ is again ill-conditioned and badly-scaled.

TABLE 6

	x_0	$10\ x_0$	$100\ x_0$
NEWTON	90, 991, 0.9(-11)	103, 1134, 0.1(-14)	91, 1002, 0.1(-13)
BROWN	66, 429, 0.4(8)	104, 676, 0.0	92, 598, 0.2(-15)
BRENT	66, 429, 0.5(8)	104, 676, 0.4(-14)	92, 598, 0.4(-14)
BRENTM	66, 429, 0.5(8)	101, 662, 0.7(-14)	89, 585, 0.7(-14)

4. Chebyquad

If T_i is the i^{th} Chebyshev polynomial shifted to the interval [0,1] then this function is defined by

$$f_k(x) = \int_0^1 T_i(s)ds - \frac{1}{n}\sum_{j=1}^{n} T_k(x_j), \qquad 1 \le k \le n . \tag{6.8}$$

The standard initial guess for this problem is

$$x_0 = (\xi_i) \quad \text{where} \quad \xi_i = i/(n+1), \quad 1 \le i \le n . \tag{6.9}$$

The results of solving this problem for n = 5 and all three starting points are given in Table 7; the results for n = 7, 8, and 9 but only for x_0 are in Table 8.

TABLE 7

	x_0	$10\ x_0$	$100\ x_0$
NEWTON	5, 31, 0.4(-11)	4, 25, 0.2(22)	6, 37, 0.4(48)
BROWN	6, 24, 0.1(-15)	39, 156, 0.1(-16)	9, 36, 0.1(22)
BRENT	5, 20, 0.1(-15)	10, 40, 0.1(-15)	16, 64, 0.1(-15)
BRENTM	3, 15, 0.1(-15)	9, 39, 0.1(-15)	14, 59, 0.2(-15)

TABLE 8

	7	8	9
NEWTON	5, 31, 0.4(-11)	4, 37, 0.4(26)	6, 61, 0.2(38)
BROWN	5, 25, 0.1(-15)	7, 39, 0.6(27)	8, 48, 0.2(-15)
BRENT	5, 25, 0.1(-15)	13, 72, 0.4(49)	6, 36, 0.3(-15)
BRENTM	3, 19, 0.1(-15)	13, 72, 0.4(49)	3, 24, 0.4(-14)

Remarks

(a) For $n = 1,\ldots,7$ and $n = 9$ this function has $n!$ zeroes $x^* = (\xi_i^*)$ with $0 \leq \xi_i^* \leq 1$, but for given n any two of these zeroes have the same components arranged in a different order.

(b) For the second starting point of $n = 5$ BROWN and BRENT converged to different solutions, and for the third starting point BROWN diverged but BRENT converged.

(c) Since (6.8) does not have a solution for $n = 8$, none of the iterations converged and this was detected by the techniques of Section 5.

5. Powell's singular function

To define this function let

$$f_1(x) = x_1 + 10x_2, \quad f_2(x) = \sqrt{5}\,(x_3-x_4)$$
$$f_3(x) = (x_2-2x_3)^2, \quad f_4(x) = \sqrt{10}\,(x_1-x_4)^2 \quad . \tag{6.10}$$

The results of solving this function are given in Table 9 where the starting point is

$$x_0 = (3,-1,0,1)$$

	x_0	$10\ x_0$	$100\ x_0$
NEWTON	18, 91, 0.1(-9)	22, 111, 0.1(-9)	25, 126, 0.6(-10)
BROWN	22, 77, 0.3(-10)	26, 91, 0.2(-10)	29, 102, 0.3(-10)
BRENT	21, 74, 0.5(-10)	25, 88, 0.2(-10)	28, 98, 0.5(-10)
BRENTM	17, 71, 0.5(-10)	21, 85, 0.2(-10)	24, 95, 0.4(-10)

Remarks

(a) The Jacobian is singular at the unique zero (the origin) of this function and therefore the usual local convergence theorems do not apply. Nevertheless convergence took place in each case.

(b) All of the algorithms in this paper are invariant under translation; that is, given F and $\hat{F}$ defined by $\hat{F}(x) = F(x-v)$ for some vector v, and initial vectors $\hat{x}_0 = x_0+v$, then the iterates will satisfy $\hat{x}_k = x_k+v$ for $k \geq 1$. However, translation affects the convergence criteria. For example, since for (6.10) the solution is at the origin, condition (6.1) was never satisfied and thus BRENTM and BRENT give the same results for (6.10). The results of Table 9 were obtained by working with $\hat{F}$ and $\hat{x}_0$ as above and $v = e_3$. This only affected the results of BRENTM.

7. CONCLUDING REMARKS

Our numerical results indicate that in general the implementation BRENT of Brent's method is superior, in terms of robustness and efficiency, to the implementation BROWN of Brown's method. They also show that the use of the iterative refinement is almost invariably desirable and thus BRENTM should be preferred over BRENT.

The above conclusions are also supported by the numerical results obtained by testing our algorithm on two other sets of test functions: The testing program of Ken Hillstrom of Argonne National Laboratory consists of 10 functions and 20 starting values per function while H. A. Watts of Sandia Laboratories has a collection of 27 functions and for each function several starting values are given.

ACKNOWLEDGEMENTS

We would like to thank Ken Hillstrom and H. A. Watts for the use of their testing programs, and Larry Nazareth for his comments on a draft of this paper.

REFERENCES

[1] Brent, R. P., "Some efficient algorithms for solving systems of nonlinear equations," *SIAM J. Numer. Anal.* 10 (1973), 327-344.

[2] Brown, K. M., "A quadratically convergent method for solving simultaneous nonlinear equations," Purdue University Ph.D. Dissertation, Lafayette, Indiana, 1966.

[3] Brown, K. M. and Dennis, J. E., "On the second order convergence of Brown's derivative-free method for solving simultaneous nonlinear equations," Department of Computer Science Technical Report 71-7, Yale University, New Haven, Conn., 1971.

[4] Cosnard, M. Y., "A comparison of four methods for solving systems of nonlinear equations," Dept. of Computer Science Technical Report 75-248, Cornell University, Ithaca, New York, 1975.

[5] Curtis, A. R., Powell, M.J.D., and Reid, J. K., "On the estimation of sparse Jacobian matrices," *J. Inst. Maths. Applics.* 13 (1974), 117-119.

[6] Gay, D. M., "Brown's method and some generalizations, with applications to minimization problems," Cornell University Ph.D. Dissertation, Ithaca, New York, 1975.

[7] Gay, D. M., "Implementing Brown's method," Center for Numerical Analysis Report CNA-109, The University of Texas at Austin, 1975.

[8] Shampine, L. F. and Gordon, M. K., "Solving systems of nonlinear equations," Sandia Laboratories Technical Report SAND-75-0450, Albuquerque, New Mexico, 1975.

APPENDIX

```
C.BRENT.....                                                          00000010
C   THIS SUBROUTINE CALCULATES AN OPTIMAL VALUE OF MOPT AND CALLS     00000020
C   BRENTM. THE PARAMETERS ARE A SUBSET OF THOSE OF BRENTM WITH THE   00000030
C   EXCEPTION OF LWA AND WA :                                         00000040
C                                                                     00000050
C       LWA IS AN INTEGER GREATER THAN OR EQUAL TO   N*N+3*N           00000060
C                                                                     00000070
C       WA IS A LINEAR ARRAY OF LENGTH LWA.                           00000080
C                                                                     00000090
C.....                                                                00000100
      SUBROUTINE BRENT(N,FCN,X,FTOL,XTOL,MAXFEV,IER,LWA,WA)           00000110
      INTEGER N,MAXFEV,IER,LWA,K,MOPT                                 00000120
      REAL *8 FTOL,XTOL,DKP1,EMAX,ONE,ZERO                            00000130
      REAL *8 X(N),WA(LWA)                                            00000140
      REAL *8 DABS,DLOG                                               00000150
      EXTERNAL FCN                                                    00000160
      DATA ZERO,ONE /0.D0,1.D0/                                       00000170
      IER = 0                                                         00000180
      IF (N .LE. 0 .OR. LWA .LT. N*N+3*N) RETURN                      00000190
      DO 10 K = 1,N                                                   00000200
         DKP1 = ONE*(K+1)                                             00000210
         WA(K) = DLOG(DKP1)/(N+2*K+1)                                 00000220
10       CONTINUE                                                     00000230
      EMAX = ZERO                                                     00000240
      DO 20 K = 1,N                                                   00000250
         IF (DABS(WA(K)) .LT. EMAX) GO TO 20                          00000260
         MOPT = K                                                     00000270
         EMAX = DABS(WA(K))                                           00000280
20       CONTINUE                                                     00000290
      CALL BRENTM(N,FCN,X,FTOL,XTOL,MAXFEV,IER,MOPT,WA(3*N+1),        00000300
     +     WA(2*N+1),WA(N+1),WA(1))                                   00000310
      RETURN                                                          00000320
      END                                                             00000330
C.BRENTM.....                                                         00000340
C                                                                     00000350
C   THIS SUBROUTINE TRIES TO FIND A ZERO TO A SYSTEM OF N SYMULTANEOUS 00000360
C   EQUATIONS IN N UNKNOWNS BY BRENT'S METHOD.                        00000370
C                                                                     00000380
C   ON INPUT:                                                         00000390
C                                                                     00000400
C       N IS THE NUMBER OF EQUATION AND UNKNOWNS.                     00000410
C                                                                     00000420
C       FCN IS THE NAME OF THE SUBROUTINE WHICH DEFINES THE SYSTEM    00000430
C         OF EQUATIONS. THE USER SPECIFIES FCN BY WRITING A SUBROUTINE 00000440
C         FCN(N,K,X,FCNK,IER) WHICH COMPUTES THE K-TH COMPONENT OF FCN 00000450
C         EVALUATED AT X AND RETURNS THE VALUE IN FCNK. IER SHOULD    00000460
C         NOT BE CHANGED UNLESS THE USER WANTS TO TERMINATE THE       00000470
C         ITERATION. IN THIS CASE SET IER TO A NEGATIVE INTEGER.      00000480
C                                                                     00000490
C       X IS AN ARAY OF LENGTH N WHICH MUST CONTAIN THE INITIAL       00000500
C         ESTIMATE TO THE ZERO OF THE SYSTEM OF EQUATIONS.            00000510
C                                                                     00000520
C       FTOL SPECIFIES THE FIRST STOPPING CRITERION. TERMINATION OCCURS 00000530
C         IF ALL THE RESIDUALS ARE LESS THAN FTOL IN MAGNITUDE.       00000540
C                                                                     00000550
C       XTOL SPECIFIES THE SECOND STOPPING CRITERION. TERMINATION     00000560
C         OCCURS IF THE RELATIVE ERROR BETWEEN TWO SUCCESSIVE ITERATES 00000570
C         IS LESS THAN XTOL.                                          00000580
C                                                                     00000590
```

```
C       MAXFEV SPECIFIES THE THIRD STOPPING CRITERION. TERMINATION      00000600
C         OCCURS IF THE NUMBER OF CALLS TO FCN EXCEEDS MAXFEV.          00000610
C                                                                       00000620
C       MOPT IS THE NUMBER OF CONSECUTIVE TIMES THAT THE APPROXIMATE    00000630
C         JACOBIAN IS REUSED DURING EACH ITERATION OF ITERATIVE         00000640
C         REFINEMENT. MAXIMUM EFFICIENCY IS USUALLY OBTAINED IF         00000650
C         MOPT MAXIMIZES LOG(K+1)/(N+2*K+1) FOR K = 1,...,N.            00000660
C                                                                       00000670
C   ON OUTPUT:                                                          00000680
C                                                                       00000690
C       X CONTAINS THE FINAL ESTIMATE FOR THE ZERO OF THE SYSTEM        00000700
C         OF EQUATIONS.                                                 00000710
C                                                                       00000720
C       MAXFEV CONTAINS THE NUMBER OF CALLS USED IN PRODUCING X.        00000730
C                                                                       00000740
C       IER IS SET AS FOLLOWS:                                          00000750
C                                                                       00000760
C         IER=0     IMPROPER INPUT PARAMETERS.                        00000770
C                                                                       00000780
C         IER=1     ABSOLUTE VALUE OF EACH RESIDUAL IS LESS THAN FTOL. 00000790
C                                                                       00000800
C         IER=2     RELATIVE ERROR BETWEEN TWO SUCCESSIVE ITERATES    00000810
C                   IS LESS THAN XTOL.                                  00000820
C                                                                       00000830
C         IER=3     CONDITIONS FOR IER=1 AND IER=2 HOLD.              00000840
C                                                                       00000850
C         IER=4     NUMBER OF CALLS TO F EXCEEDS MAXFEV.              00000860
C                                                                       00000870
C         IER=5     APPROXIMATE JACOBIAN MATRIX IS SINGULAR.          00000880
C                                                                       00000890
C         IER=6     ITERATION IS NOT MAKING GOOD PROGRESS.            00000900
C                                                                       00000910
C         IER=7     ITERATION IS DIVERGING.                           00000920
C                                                                       00000930
C         IER=8     ITERATION SEEMS TO BE CONVERGING BUT THE REQUESTED 00000940
C                   ACCURACY IS TOO STRINGENT, OR THE CONVERGENCE      00000950
C                   IS VERY SLOW DUE TO A JACOBIAN SINGULAR NEAR      00000960
C                   THE ITERATES OR DUE TO BADLY SCALED VARIABLES.    00000970
C                                                                       00000980
C   WORKING ARRAYS:                                                     00000990
C                                                                       00001000
C       Q IS AN N BY N ARRAY.                                           00001010
C       Y,Z,A ARE LINEAR ARAYS OF LENGTH N.                             00001020
C                                                                       00001030
C.....................................................................  00001040
      SUBROUTINE BRENTM(N,FCN,X,FTOL,XTOL,MAXFEV,IER,MOPT,Q,Y,Z,A)      00001050
      INTEGER I,IER,J,K,M,MAXFEV,MOPT,N,NFEVAL,NIER6,NIER7,NIER8        00001060
      REAL *8 DIFIT,DIFIT1,EPS,EPSMCH,ETA,ETA1,FKZ,FNORM,FNORM1,        00001070
     +        FONCT,FTOL,H,ONE,P05,RC,SIGMA,XNORM,XTOL,ZERO             00001080
      REAL *8 A(N),Q(N,N),X(N),Y(N),Z(N)                                00001090
      REAL *8 DMAX1,DABS,DSQRT                                          00001100
      LOGICAL CONV,SING                                                 00001110
      DATA ZERO,ONE,P05 /0.D0,1.D0,5.D-2/                               00001120
      IER = 0                                                           00001130
      IF (N .LE. 0) RETURN                                              00001140
C                                                                       00001150
C     EPSMCH IS THE MACHINE PRECISION.                                  00001160
C                                                                       00001170
      EPSMCH = 16.D0**(-13)                                             00001180
```

```
      EPS = DSQRT(EPSMCH)                                               00001190
C                                                                       00001200
C     NFEVAL COUNTS THE NUMBER OF COMPONENT FUNCTION EVALUATIONS.       00001210
C     XNORM IS THE NORM OF X.                                           00001220
C     FNORM IS THE COMPUTED RESIDUAL WITH MAXIMUM ABSOLUTE VALUE.       00001230
C     DIFIT IS THE NORM OF THE DIFFERENCE BETWEEN THE LAST TWO ITERATES.00001240
C     NIER6 IS USED TO DETERMINE WHEN TO SET IER=6.                     00001250
C     NIER7 IS USED TO DETERMINE WHEN TO SET IER=7.                     00001260
C     NIER8 IS USED TO DETERMINE WHEN TO SET IER=8.                     00001270
C                                                                       00001280
      NFEVAL = 0                                                        00001290
      XNORM = ZERO                                                      00001300
      FNORM = ZERO                                                      00001310
      DIFIT = ZERO                                                      00001320
      NIER6 = -1                                                        00001330
      NIER7 = -1                                                        00001340
      NIER8 = 0                                                         00001350
      DO 10 I = 1,N                                                     00001360
         XNORM = DMAX1(XNORM,DABS(X(I)))                                00001370
10       CONTINUE                                                       00001380
C                                                                       00001390
C     ENTER THE PRINCIPAL ITERATION.                                    00001400
C                                                                       00001410
20    CONTINUE                                                          00001420
      SING = .TRUE.                                                     00001430
      FNORM1 = FNORM                                                    00001440
      FNORM = ZERO                                                      00001450
      DIFIT1 = DIFIT                                                    00001460
C                                                                       00001470
C     COMPUTE THE STEP H FOR THE DIVIDED DIFFERENCE WHICH               00001480
C     APPROXIMATES THE K-TH ROW OF THE JACOBIAN MATRIX.                 00001490
C                                                                       00001500
      H = EPS*DMAX1(XNORM,ONE)                                          00001510
      DO 40 J = 1,N                                                     00001520
         DO 30 I = 1,N                                                  00001530
            Q(I,J) = ZERO                                               00001540
30          CONTINUE                                                    00001550
         Q(J,J) = H                                                     00001560
         Y(J) = X(J)                                                    00001570
40       CONTINUE                                                       00001580
C                                                                       00001590
C     ENTER A SUBITERATION.                                             00001600
C                                                                       00001610
      DO 130 K = 1,N                                                    00001620
         CALL FCN(N,K,Y,FONCT,IER)                                      00001630
         IF (IER .LT. 0) GO TO 210                                      00001640
         NFEVAL = NFEVAL+1                                              00001650
         FNORM = DMAX1(FNORM,DABS(FONCT))                               00001660
C                                                                       00001670
C        COMPUTE THE K-TH ROW OF THE JACOBIAN MATRIX.                   00001680
C                                                                       00001690
         DO 60 J = K,N                                                  00001700
            DO 50 I = 1,N                                               00001710
               Z(I) = Y(I)+Q(I,J)                                       00001720
50             CONTINUE                                                 00001730
            CALL FCN(N,K,Z,FKZ,IER)                                     00001740
            IF (IER .LT. 0) GO TO 210                                   00001750
            NFEVAL = NFEVAL+1                                           00001760
            A(J) = FKZ-FONCT                                            00001770
```

```
60          CONTINUE                                                    00001780
C                                                                       00001790
C       COMPUTE THE HOUSEHOLDER TRANSFORMATION TO REDUCE THE K-TH ROW   00001800
C       OF THE JACOBIAN MATRIX TO A MULTIPLE OF THE K-TH UNIT VECTOR.   00001810
C                                                                       00001820
        ETA = ZERO                                                      00001830
        DO 70 I = K,N                                                   00001840
           ETA = DMAX1(ETA,DABS(A(I)))                                  00001850
70         CONTINUE                                                     00001860
        IF (ETA .EQ. ZERO) GO TO 130                                    00001870
        SING = .FALSE.                                                  00001880
        SIGMA = ZERO                                                    00001890
        ETA1 = ONE/ETA                                                  00001900
        DO 80 I = K,N                                                   00001910
           A(I) = A(I)*ETA1                                             00001920
           SIGMA = SIGMA+A(I)*A(I)                                      00001930
80         CONTINUE                                                     00001940
        SIGMA = DSQRT(SIGMA)                                            00001950
        IF (A(K) .LT. ZERO) SIGMA = -SIGMA                              00001960
        A(K) = A(K)+SIGMA                                               00001970
C                                                                       00001980
C       APPLY THE TRANSFORMATION AND COMPUTE THE ORTHOGONAL MATRIX Q.   00001990
C                                                                       00002000
        DO 110 I = 1,N                                                  00002010
           RO = ZERO                                                    00002020
           DO 90 J = K,N                                                00002030
              RO = RO+A(J)*Q(I,J)                                       00002040
90            CONTINUE                                                 00002050
           RO = RO/(SIGMA*A(K))                                         00002060
           DO 100 J = K,N                                               00002070
              Q(I,J) = Q(I,J)-RO*A(J)                                   00002080
100           CONTINUE                                                 00002090
110        CONTINUE                                                     00002100
C                                                                       00002110
C       COMPUTE THE NEW SUBITERATE.                                     00002120
C                                                                       00002130
        A(K) = SIGMA*ETA                                                00002140
        SIGMA = FONCT/A(K)                                              00002150
        DO 120 I = 1,N                                                  00002160
           Y(I) = Y(I)+SIGMA*Q(I,K)                                     00002170
120        CONTINUE                                                     00002180
130     CONTINUE                                                        00002190
C                                                                       00002200
C     COMPUTE THE NORM OF THE ITERATE AND THE NORM OF THE DIFFERENCE    00002210
C     BETWEEN THE LAST TWO ITERATES.                                    00002220
C                                                                       00002230
      XNORM = ZERO                                                      00002240
      DIFIT = ZERO                                                      00002250
      DO 140 I = 1,N                                                    00002260
         XNORM = DMAX1(XNORM,DABS(Y(I)))                                00002270
         DIFIT = DMAX1(DIFIT,DABS(X(I)-Y(I)))                           00002280
         X(I) = Y(I)                                                    00002290
140      CONTINUE                                                       00002300
C                                                                       00002310
C     DETERMINE THE PROGRESS OF THE ITERATION.                          00002320
C                                                                       00002330
      CONV = .FALSE.                                                    00002340
      IF (FNORM .LT. FNORM1 .AND. DIFIT .LT. DIFIT1) CONV = .TRUE.      00002350
      NIER6 = NIER6+1                                                   00002360
```

```
      NIER7 = NIER7+1                                                    00002370
      NIER8 = NIER8+1                                                    00002380
      IF (CONV) NIER6=0                                                  00002390
      IF (FNORM .LT. FNORM1 .OR. DIFIT .LT. DIFIT1) NIER7=0              00002400
      IF (FNORM .GT. EPS .AND. DIFIT .GT. EPS*DMAX1(XNORM,ONE)) NIER8=0  00002410
C                                                                        00002420
C     STOPPING CRITERIA.                                                 00002430
C                                                                        00002440
      IF (FNORM .LT. FTOL) IER=1                                         00002450
      IF (DIFIT .LT. XTOL*XNORM .AND. CONV) IER=2                        00002460
      IF (FNORM .LT. FTOL .AND. IER .EQ. 2) IER=3                        00002470
      IF (NFEVAL .GT. MAXFEV) IER=4                                      00002480
      IF (SING) IER=5                                                    00002490
      IF (NIER6 .GE. 5) IER=6                                            00002500
      IF (NIER7 .GE. 3) IER=7                                            00002510
      IF (NIER8 .GE. 4) IER=8                                            00002520
      IF (IER .NE. 0) GO TO 210                                          00002530
C                                                                        00002540
C     ITERATIVE REFINEMENT IS ONLY USED IF THE ITERATION IS CONVERGING.  00002550
C                                                                        00002560
      IF (.NOT. CONV .OR. DIFIT .GE. P05*XNORM) GO TO 200                00002570
C                                                                        00002580
C     START ITERATIVE REFINEMENT.                                        00002590
C                                                                        00002600
      IF (MOPT .EQ. 1) GO TO 200                                         00002610
      DO 190 M = 2,MOPT                                                  00002620
         FNORM1 = FNORM                                                  00002630
         FNORM = ZERO                                                    00002640
         DO 170 K = 1,N                                                  00002650
            CALL FCN(N,K,Y,FONCT,IER)                                    00002660
            IF (IER .LT. 0) GO TO 210                                    00002670
            NFEVAL = NFEVAL+1                                            00002680
            FNORM = DMAX1(FNORM,DABS(FONCT))                             00002690
C                                                                        00002700
C           ITERATIVE REFINEMENT IS TERMINATED IF IT DOES NOT            00002710
C           GIVE A PEDUCTION OF THE RESIDUALS OR IF A(K) IS ZERO.        00002720
C                                                                        00002730
            IF (FNORM .LT. FNORM1 .AND. A(K) .NE. ZERO) GO TO 150        00002740
            FNORM = FNORM1                                               00002750
            GO TO 200                                                    00002760
150         CONTINUE                                                     00002770
            SIGMA = FONCT/A(K)                                           00002780
            DO 160 I = 1,N                                               00002790
               Y(I) = Y(I)+SIGMA*Q(I,K)                                  00002800
160            CONTINUE                                                  00002810
170         CONTINUE                                                     00002820
C                                                                        00002830
C        COMPUTE THE NORM OF THE ITERATE AND THE NORM OF THE DIFFERENCE  00002840
C        BETWEEN THE LAST TWO ITERATES OF THE ITERATIVE REFINEMENT.      00002850
C                                                                        00002860
         XNORM = ZERO                                                    00002870
         DIFIT = ZERO                                                    00002880
         DO 180 I = 1,N                                                  00002890
            XNORM = DMAX1(XNORM,DABS(Y(I)))                              00002900
            DIFIT = DMAX1(DIFIT,DABS(X(I)-Y(I)))                         00002910
            X(I) = Y(I)                                                  00002920
180         CONTINUE                                                     00002930
C                                                                        00002940
C        STOPPING CRITERIA FOR ITERATIVE REFINEMENT.                     00002950
```

```
C                                                                       00002960
      IF (FNORM .LT. FTOL) IER=1                                        00002970
      IF (DIFIT .LT. XTOL*XNORM .AND. CONV) IER=2                       00002980
      IF (FNORM .LT. FTOL .AND. IER .EQ. 2) IER=3                       00002990
      IF (NFEVAL .GT. MAXFEV) IER=4                                     00003000
      IF (IER .NE. 0) GO TO 210                                         00003010
190   CONTINUE                                                          00003020
200   CONTINUE                                                          00003030
C                                                                       00003040
C     END OF THE ITERATIVE REFINEMENT.                                  00003050
C                                                                       00003060
      GO TO 20                                                          00003070
210   CONTINUE                                                          00003080
      MAXFEV = NFEVAL                                                   00003090
      RETURN                                                            00003100
      END                                                               00003110
```

```
C                                                                       00002960
         IF (FNORM .LT. FTOL) IER=1                                     00002970
         IF (DIFIT .LT. XTOL*XNORM .AND. CONV) IER=2                    00002980
         IF (FNORM .LT. FTOL .AND. IER .EQ. 2) IER=3                    00002990
         IF (NFEVAL .GT. MAXFEV) IER=4                                  00003000
         IF (IER .NE. 0) GO TO 210                                      00003010
190      CONTINUE                                                       00003020
200   CONTINUE                                                          00003030
C                                                                       00003040
C     END OF THE ITERATIVE REFINEMENT.                                  00003050
C                                                                       00003060
      GO TO 20                                                          00003070
210   CONTINUE                                                          00003080
      MAXFEV = NFEVAL                                                   00003090
      RETURN                                                            00003100
      END                                                               00003110
```