A QUASI-NEWTON METHOD FOR SOLVING SMALL NONLINEAR SYSTEMS OF ALGEBRAIC EQUATIONS

JEFFREY L. MARTIN

Abstract. Finding roots of nonlinear algebraic systems is paramount to many applications in applied mathematics because the physical laws of interest are usually nonlinear. If the functions comprising a system are complicated it is beneficial to reduce computing the functions as much as possible. The following describes a quasi-Newton root solver called Broyden Approximate Norm Descent, or 'BroydenAND', which can employ either Broyden's first or second methods, and is shown to be overall less expensive in terms of function evaluations than several competing root solvers. The results for Broyden's first and second methods are identical for all but the last test function, for which the two methods start to diverge in the presence of round-off errors.

Key words. Broyden's method, nonlinear system, backtracking line search, Sherman-Morrison, superlinear

AMS subject classifications. 65B04, 65F04, 65H04, 65H10

1. A Brief History of Newton's Method. The historical road to Newton's method is interesing in its own right, setting aside the technical details. Sir Isaac Newton did indeed play a role in its development, but there were certainly other key players. Despite the uncertainties, most experts agree that the road begins with a perturbation technique for solving scalar polynomial equations, pioneered by Francois Vieta in 1600 (Deuflhard [3]). This method used one decimal place of the computed solution on each step, and converged linearly [3]. In 1664 Isaac Newton learned of Vieta's method, and by 1669 he had improved it by linearizing the resulting equations [3]. One of his first applications of this newfangled scheme was on the numerical solution of the cubic polynomial equation

$$f(x) = x^3 - 2x - 5 = 0 \tag{1.1}$$

By using an initial guess x_0 of 2, he created an iterative method by substituting $x_k = x_{k-1} + \delta$ into equation (1.1) and solving for δ (neglecting terms higher than first order), and repeating, etc, which produced a better and better approximation to the true root [3]. Newton realized that by keeping all decimal places of the corrections, the number of accurate places would double, which means quadratic convergence. However, there is no evidence that Newton incorporated derivatives in his method. In 1690, Joseph Raphson improved Newton's method of 1669 by making the scheme fully iterative, making computation of successive polynomials unnecessary [3]. But it was in 1740 when the derivative aspect of "Newton's Method" was brought to the fore by Thomas Simpson [3]. Simpson extended Newton's formulation from 1669 by introducing derivatives, and wrote the mathematical formula for the iterative method for both a scalar nonlinear equation, and a system of two equations and two unknowns [3]. For a scalar equation and some initial guess x_0, this formulation was given by equation 1.2.

$$x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})} \qquad for \;\; k = 1, 2, ... \tag{1.2}$$

Thus, it was Simpson who made the leap from using this procedure on scalar equations to systems of equations.

2. An Old Idea: Secant Updating. Variations of what is now known as the secant method date back to the Babylonian era some 4000 years ago, for there is evidence that they used the secant updating formula to solve equations in one dimension (Griewank [5]). This requires two initial guesses x_0 and x_1 and computes the updated solution as in equation (2.1).

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} \quad for \;\; k = 1, 2, ... \tag{2.1}$$

In that time, the conception of differential calculus was almost certainly unknown. Today, derivatives are understood very well, but analytical evaluation of derivatives can be exorbitant if the underlying functions themselves are complicated. This was in fact a problem for Charles Broyden in the 1960's, when he was working with nonlinear reactor models for the English Electric Company in Leicester [5]. When trying to solve nonlinear systems that arose from the discretization of these models, he realized that instead of repeatedly evaluating and factoring the Jacobian, he could use secant information (function value differences and the previous solution information) to directly compute an approximation to the Jacobian [5]. In the one-dimensional case, the problem is trivial because a unique approximation, B_k, can be computed by dividing the function difference by the solution difference. Thus, letting $\Delta_k = F_k - F_{k-1}$ and $\delta_k = x_k - x_{k-1}$,

$$B_k = \frac{\Delta_k}{\delta_k} \approx F'(x_k) \tag{2.2}$$

However, for multi-dimensional systems, the secant condition provides n conditions for the n^2 degrees of freedom in the new Jacobian approximation (Kelley [10]). Broyden's stroke of genius was realizing that the Jacobian approximation from the previous iteration, B_{k-1}, could be updated with a certain rank-one update. This update would satisfy not only the secant condition, but also the condition that $B_k w = B_{k-1} w$ for all directions $w \in R^n$ orthogonal to δ_k (i.e. $w_k^T \delta_k = 0$) [5]. This idea led Broyden to develop three methods that use secant information in lieu of derivative information to solve nonlinear systems (two of them are implemented in this paper). Broyden published these results in his 1965 paper (see Broyden [1]).

3. Newton vs. Quasi-Newton. Systems of equations arise so frequently in not just pure mathematics, but also in the physical science and engineering disciplines. The physical laws that direct mathematicians' and engineers' research can almost always be transformed from a continuous system to a discrete system of equations, and more often than not, they are nonlinear. In fact, nonlinear systems abound in mathematical problems. This makes for the phenomena being modeled more diverse and interesting, but, the price to pay is that finding solutions is nontrivial. Then how are systems of nonlinear equations ultimately solved, and do the solvers always succeed?

The two main class of methods used for solving nonlinear algebraic equations are Newton and quasi-Newton. Equivalently, it could be said that the two class of methods are derivative and derivative-free, respectively. The problem of solving a nonlinear algebraic system is equivalent to the statement

$$For \; F : R^n \to R^n, find \; \mathbf{x}^* \in R^n \; such \; that \; F(\mathbf{x}^*) = 0 \tag{3.1}$$

where F($\mathbf{x}$) is some nonlinear vector-valued function with n arguments,

$$F(x_1, x_2, ..., x_n) = (f_1, f_2, ..., f_n) \tag{3.2}$$

Performing a Taylor expansion of F about some $\mathbf{x_0}$

$$F(\mathbf{x}) = F(\mathbf{x_0}) + J(\mathbf{x_0}) \cdot (\mathbf{x} - \mathbf{x_0}) + O(\mathbf{x} - \mathbf{x_0})^2 \tag{3.3}$$

Let J denote the exact Jacobian matrix associated with F. Newton's method is based on a linear approximation, so the $O(h^2)$ and higher order terms are neglected. If x* is the root, then substituting this into equation (3.3),

$$0 \approx F(\mathbf{x_0}) + J(\mathbf{x_0}) \cdot (\mathbf{x}^* - \mathbf{x_0}) \tag{3.4}$$

$$\mathbf{x}^* \approx \mathbf{x_0} - J(\mathbf{x_0})^{-1} F(\mathbf{x_0}) \tag{3.5}$$

Equation (3.5) can be turned into an iterative process, with some initial vector $\mathbf{x_0}$,

$$\mathbf{x_k} = \mathbf{x_{k-1}} - J(\mathbf{x_{k-1}})^{-1} F(\mathbf{x_{k-1}}), \quad k = 1, 2, ... \tag{3.6}$$

To avoid taking the inverse, one could write

$$\begin{aligned} J(\mathbf{x_{k-1}})\mathbf{s_k} &= -F(\mathbf{x_{k-1}}) \\ \mathbf{x_k} &= \mathbf{x_{k-1}} + \mathbf{s_k} \end{aligned} \tag{3.7}$$

and perform a linear solve to get s_k. If an initial vector $\mathbf{x_0}$ is chosen that is sufficiently close to the true root $\mathbf{x}^*$ and $J(\mathbf{x}^*)$ is nonsingular, then Newton's method converges Q-superlinearly to $\mathbf{x}^*$ (Martinez [13]), meaning

$$\lim_{k \to \infty} \frac{||\mathbf{x_k} - \mathbf{x}^*||}{||\mathbf{x_{k-1}} - \mathbf{x}^*||} = 0 \tag{3.8}$$

Additionally, if $J(\mathbf{x})$ is Lipschitz condinuous for all $\mathbf{x}$ close enough to $\mathbf{x}^*$, for some constant L, meaning

$$||J(\mathbf{x}) - J(\mathbf{x}^*)|| \leq L||\mathbf{x} - \mathbf{x}^*|| \tag{3.9}$$

then Newton's method enjoys quadratic convergence, meaning the error at iteration k is proportional to the square of the error at iteration k-1 [13]. These theoretical properties make Newton's method an ideal choice. However, If the vector-valued function F is complicated and expensive to evaluate, then the Jacobian will be equally or more expensive to evaluate. This motivates a method that utilizes a matrix, B_k, that is an approximation to the true Jacobian. Then equation (3.6) is re-written as

$$\mathbf{x_k} = \mathbf{x_{k-1}} - B_{k-1}^{-1} F(\mathbf{x_{k-1}}) \tag{3.10}$$

With each successive iteration, the goal is to force B_k to become closer to the true Jacobian, and thus also force x_k to become closer to x^*. The class of methods written in the form of equation (3.10) are called quasi-Newton methods, and in general converge Q-superlinearly. Interestingly, most of the known quasi-Newton methods do not require consistency (i.e. that the sequence $\{B_k\}$ converge to the Jacobian matrix at the root) for convergence, as posited by Dennis et al. [7]. There are many types of quasi-Newton methods, but Broyden's methods are among the most popular.

4. Broyden: the Good and the Bad.

4.1. Broyden's first method. A fundamental challenge in any quasi-Newton method is finding a stable and efficient method for updating the Jacobian approximation, B_{k-1}. In 1965, Charles Broyden expounded two methods for accomplishing this task (Broyden [1]). The first method updates B_{k-1} directly, and does a linear solve on each iteration to obtain a new solution, x_k. To gain a sense of how this method was developed, consider a linear approximation of F($\mathbf{x}$) about some $\mathbf{x_k}$,

$$F(\mathbf{x}) \approx F(\mathbf{x_k}) + J(\mathbf{x_k}) \cdot (\mathbf{x} - \mathbf{x_k}) \tag{4.1}$$

If B_k is the best approximation available to the true Jacobian $J(\mathbf{x_k})$, then one should expect that

$$F(\mathbf{x_{k-1}}) \approx F(\mathbf{x_k}) + B_k \cdot (\mathbf{x_{k-1}} - \mathbf{x_k}) \tag{4.2}$$

Let $\delta_k = \mathbf{x_k} - \mathbf{x_{k-1}}$, and $\Delta_k = F(\mathbf{x_k}) - F(\mathbf{x_{k-1}})$, and equation (4.2) becomes

$$B_k \delta_\mathbf{k} \approx \Delta_k \tag{4.3}$$

Equation (4.3) is called the quasi-Newton condition (also referred to as the secant relation) (Sauer [16]). However, for the multivariable case (i.e. n $>$ 1), the quasi-Newton condition does not prescribe a unique B_k [16]. In order to determine a unique B_k, Broyden posited that since there is no new information obtained from the orthogonal complement of $\delta_\mathbf{k}$, then there is no reason why B_{k-1} would change in a direction $\mathbf{w}$, where ${\delta_\mathbf{k}}^T\mathbf{w} = 0$ [16]. So rather than computing B_k fom scratch, the previous approximation B_{k-1} should be updated such that both the secant condition and equation (4.4) is satisfied.

$$B_k\mathbf{w} = B_{k-1}\mathbf{w} \tag{4.4}$$

An approximate Jacobian satisfying both (4.3) and (4.4) is

$$B_k = B_{k-1} + \frac{(\Delta_k - B_{k-1}\delta_\mathbf{k}){\delta_\mathbf{k}}^T}{{\delta_\mathbf{k}}^T\delta_\mathbf{k}} \tag{4.5}$$

To verify equation (4.5), one can plug it in to (4.3) and (4.4). So (4.3) becomes

$$B_k\delta_\mathbf{k} = B_{k-1}\delta_\mathbf{k} + \frac{(\Delta_k - B_{k-1}\delta_\mathbf{k}){\delta_\mathbf{k}}^T\delta_\mathbf{k}}{{\delta_\mathbf{k}}^T\delta_\mathbf{k}} = \Delta_k \tag{4.6}$$

and equation (4.4) becomes

$$B_k\mathbf{w} = B_{k-1}\mathbf{w} + \frac{(\Delta_k - B_{k-1}\delta_\mathbf{k}){\delta_\mathbf{k}}^T\mathbf{w}}{{\delta_\mathbf{k}}^T\delta_\mathbf{k}} = B_{k-1}\mathbf{w} \tag{4.7}$$

because ${\delta_\mathbf{k}}^T\mathbf{w} = 0$. Equation (4.5) is called Broyden's first method; it is the minimum change to B_{k-1} that satisfies the quasi-Newton condition (Griewank [5]). With these primitive ideas, a simple implementation of Broyden's first method to solve $F(\mathbf{x}) = 0$ is given in algorithm 1, assuming some initial data $\mathbf{x_0}$ and the function $F(\mathbf{x})$ are given. The initial Jacobian approximation B_0 could be set to the identity matrix if one so chooses [16].

Algorithm 1 Broyden 1

$B_0 = \mathbf{I}$ ▷ Initialize B
for k=1,2,3,... **do**
$\delta_k = -B_{k-1} \backslash F(\mathbf{x_{k-1}})$ ▷ "\" denotes the backslash command for the linear solve
$F(\mathbf{x_k}) = F(\mathbf{x_{k-1}} + \delta_k)$
$\Delta_k = F(\mathbf{x_k}) - F(\mathbf{x_{k-1}})$
$B_k = B_{k-1} + \frac{(\Delta_k - B_{k-1}\delta_\mathbf{k}){\delta_\mathbf{k}}^T}{{\delta_\mathbf{k}}^T \delta_\mathbf{k}}$ ▷ Perform the Broyden update
end for

4.2. Broyden's second method. The motivation for Broyden's second method is an effort to eliminate the requirement of a linear solve to compute the step direction (Sauer [16]). Recall that LU factorization requires $O(n^3)$ operations. But upon inspection of algorithm 1, only B_{k-1}^{-1} is directly needed to compute the step direction, and not B_{k-1}. So by replacing the update on B_{k-1} in Broyden's 1st method with an update on B_{k-1}^{-1}, the linear solve to get the step direction can be avoided. This can be accomplished by writing the Broyden update as a rank-1 update [16]. Let

$$u = \frac{(\Delta_k - B_{k-1}\delta_k)}{\delta_k^T \delta_k} \tag{4.8}$$

and let v = δ_k. Then the Broyden update can be written as the sum of B_{k-1} and the outer product uv^T.

$$B_k = B_{k-1} + uv^T \tag{4.9}$$

The linear solve at step k+1 then has the form

$$(B_{k-1} + uv^T)\delta_{k+1} = -F(\mathbf{x_k}) \tag{4.10}$$

Premultiplying (4.10) by B_{k-1}^{-1}, letting $m = B_{k-1}^{-1}\delta_{k+1}$, and letting $n = -B_{k-1}^{-1}F(\mathbf{x_k})$, equation (4.10) becomes

$$\delta_{k+1} + mv^T\delta_{k+1} = n \tag{4.11}$$

Premultiplying (4.11) by v^T and letting $v^T\delta_{k+1} = \phi$, equation (4.11) becomes

$$\phi(1 + v^T m) = v^T n \tag{4.12}$$

Solving for ϕ, we get

$$\phi = \frac{v^T n}{1 + v^T m} \tag{4.13}$$

So equation (4.11) becomes

$$\delta_{k+1} = n - \phi m = n - m\phi \tag{4.14}$$

because ϕ is a scalar. Substituting back again for ϕ, m, and n, equation (4.14) becomes

$$\delta_{k+1} = (B_{k-1}^{-1} - \frac{B_{k-1}^{-1}uv^T B_{k-1}^{-1}}{1 + v^T B_{k-1}^{-1}u})(-F(\mathbf{x_k})) \tag{4.15}$$

See (Leung [11]) for this Sherman-Morrison derivation. Equation (4.15) holds as long as $v^T B_{k-1}^{-1} u \neq -1$. Therefore, to solve for δ_{k+1}, no linear solve is required, and equation (4.15) reveals that

$$B_k^{-1} = B_{k-1}^{-1} - \frac{B_{k-1}^{-1} u v^T B_{k-1}^{-1}}{1 + v^T B_{k-1}^{-1} u} \tag{4.16}$$

This can be used as an iterative process similar to algorithm 1, but instead of updating B_{k-1}, updating B_{k-1}^{-1}. Now substituting back for u and v, equation (4.16) becomes

$$B_k^{-1} = B_{k-1}^{-1} + \frac{(\delta_k - B_{k-1}^{-1}\Delta_k)\delta_k^T B_{k-1}^{-1}}{\delta_k^T B_{k-1}^{-1} \Delta_k} \tag{4.17}$$

Equation (4.17) is a special case of a more general formula called the Sherman-Morrison-Woodbury formula, which is the generalization to a rank-n modification of B_{k-1} (Deng [2]).

Algorithm 2 Broyden 2

$B_0 = \mathbf{I}$
for k=1,2,3,... **do**
  $\delta_k = -B_{k-1} \cdot F(\mathbf{x_{k-1}})$
  $F(\mathbf{x_k}) = F(\mathbf{x_{k-1}} + \delta_k)$
  $\Delta_k = F(\mathbf{x_k}) - F(\mathbf{x_{k-1}})$
  $B_k = B_{k-1} + \frac{(\delta_k - B_{k-1}\Delta_k)\delta_k^T B_{k-1}}{\delta_k^T B_{k-1} \Delta_k}$
end for

A simple implementation of Broyden's 2nd method is given in algorithm 2. The Sherman-Morrison update requires $O(n^2)$ operations, as opposed to the $O(n^3)$ operations of the linear solve. One disadvantage of Broyden 2 is that estimates for the Jacobian, which may be required for some applications, are not readily available. This is why in some communities, Broyden 1 and 2 are referred to as "Good Broyden" and "Bad Broyden", respectively [16].

4.3. A third method. A brief word will be said about Broyden's third method, mentioned in his paper (Broyden [1]). The first two methods are useful for solving general nonlinear systems, without making any strict assumptions about the functions comprising the system. But if the functions are the first partial derivatives of a convex function, then solving the system is tantamount to minimizing the function [1]. Thus, the Jacobian matrix is symmetric positive definite at the solution, and B_k can be chosen to meet these requirements, as in equation (4.18) [1].

$$B_k = B_{k-1} - \frac{B_{k-1}\Delta_k \Delta_k^T B_{k-1}}{\Delta_k^T B_{k-1} \Delta_k} - \frac{\delta_k \delta_k^T}{\delta_k^T \Delta_k} \tag{4.18}$$

4.4. Convergence properties. The convergence of Broyden's methods is usually not as good as that of Newton's method (Griewank [4]). Generally speaking, if the initial guess $\mathbf{x_0}$ is "sufficiently close" to the root, and the initial Jacobian approximation B_0 is sufficiently close to the exact Jacobian, then Broyden's methods converge Q-superlinearly [4]. Putting this statement in mathematical form, if there exists some ϵ and ϵ_B such that $||\mathbf{x_0} - \mathbf{x}^*|| < \epsilon$ and $||B_0 - J(\mathbf{x}^*)|| < \epsilon_B$, then it converges as

$$\lim_{k \to \infty} \frac{||\mathbf{x_k} - \mathbf{x}^*||}{||\mathbf{x_{k-1}} - \mathbf{x}^*||} = 0 \tag{4.19}$$

However, Broyden's methods can sometimes do better. As x_k approaches the solution, the assumptions made in deriving equation (3.6) become more valid. So then if B_k approaches the true Jacobian, the rate of convergence will eventually become quadratic (Broyden [1]). Equation 4.19 also hints that Broyden's methods are not guaranteed to converge, and this is indeed the case. Generally speaking, the initial estimates for the solution and the Jacobian must be sufficiently close to the true ones. Exactly how close they have to be depends on the problem at hand.

5. Step Size Control and Better Jacobian Estimates.

5.1. Backtracking line search. In practice, algorithms 1 and 2 are rather naive. For most nonlinear problems, the step direction, δ_k, must be controlled with some parameter λ_i, called the step size. For this, a derivative-free line search must be employed. The purpose of a line search is to modify a hitherto locally convergent scheme and make it globally convergent (Li et al. [12]). An effective line search controls the value of λ_i in order to ensure that the root solver is pushing $F(\mathbf{x})$ "downhill". The notion of downhill in this context can be defined as the condition stating $\|F(\mathbf{x_{k-1}} + \lambda\delta_k)\| \leq$ some function of $\|F(\mathbf{x_{k-1}})\|$ and $\|\delta_k\|$. The line search employed here is an approximate norm descent line search [12], with the condition given by (5.1)

$$\|F(\mathbf{x} + \lambda\delta_k)\| \leq (1 + \eta_k)\|F(\mathbf{x_k})\| - \sigma_1\|\lambda\delta_k\|^2 \tag{5.1}$$

where σ_1 is some constant > 0 and $\{\eta_k\}$ is a positive sequence that satisfies, for some $\eta > 0$,

$$\sum_{k=0}^{\infty} \eta_k \leq \eta < \infty \tag{5.2}$$

The procedure for enforcing (5.1) is to use a backtracking process, where the algorithm checks if the condition is satisfied. If not, then λ_i is reduced to some λ_{i+1}, and so on and so forth, until the condition is satisfied. Since there is no gaurantee that the condition will be satisfied, this loop will only run for a specified number of iterations before it terminates and reports a line search failure. If the line search is unsuccessful, BroydenAND accepts the most recent step-size tested from the line search but instead of doing a Broyden update, it computes a finite difference approximation of the Jacobian, using the solution from the previous iteration. The finite difference scheme used is discussed in more detail in the following section. The theoretical framework for (5.1) is beyond the scope of this report, but it is worth mentioning the important points. If one wants to put together a line search, the most obvious choice may be to force the condition

$$\|F(\mathbf{x_{k-1}} + \lambda\delta_k)\| < \|F(\mathbf{x_{k-1}})\| \tag{5.3}$$

This condition is enforced by the derivative-free line search proposed by Griewank [4], as described below. Let

$$q_{k-1}(\lambda) = \frac{F(\mathbf{x_{k-1}})^T(F(\mathbf{x_{k-1}}) - F(\mathbf{x_{k-1}} + \lambda\delta_k))}{\|F(\mathbf{x_{k-1}}) - F(\mathbf{x_{k-1}} + \lambda\delta_k)\|^2} \tag{5.4}$$

Then the requirement for λ is that for some $\epsilon \in (0, 1/6)$,

$$q_{k-1}(\lambda) \geq \frac{1}{2} + \epsilon \tag{5.5}$$

And condition (5.3) follows from (5.5) [12]. Li and Fukushima point out that from (5.4), if $F(\mathbf{x_{k-1}})^T \nabla F(\mathbf{x_{k-1}})\delta_k = 0$, then there may not be any λ satisfying (5.5). Algorithm 3 in appendix A shows the implementation of BroydenAND with Broyden's first method, with the approximate norm descent line search. The step direction, δ_k is computed with a linear solve on line 6, using the backslash command in Matlab. Since B_{k-1} is square, the backslash operator performs PA=LU factorization, and solves by back substitution. The check on line 7 is to determine whether a line search needs to be performed [12]. The parameter $\rho \in (0, 1)$ and σ_2 is some constant > 0 [12]. If this condition is not met, the code goes to line 13 to prepare for the line search. The step size λ is continuously reduced by multiplying with some parameter τ, where $0 < \tau < 1$, until the condition on line 15 is met. The parameter σ_1 is some constant > 0 [12]. To prevent the step size from becoming too small, the while loop that reduces λ will only iterate while i < imax, where imax is specified by the user. If i reaches imax, then an error message is printed out, and the code breaks out of the for loop. If a suitable value of λ is found, then the Broyden update is performed on line 30. The algorithm checks if the error (2-norm of $F(\mathbf{x_k})$) is less than the prescribed tolerance on line 37. If it is, the program returns, and if not, the for loop repeats until it reaches the specified maximum iteration count.

The full implementation of Broyden's second method is almost identical to algorithm 3, with two obvious changes. Line 6 wouldn't be a linear solve, but rather, a multiplication. Line 30 would also change because the Jacobian approximation update is the Sherman-Morrison update.

5.2. Other step control strategies. The line search strategy used in this project, taken from Li et al. [12], is certainly not the only one. For convergence of Broyden's methods, C.T. Kelley suggests that the step size, λ should satisfy the Armijo rule when a line search is used, which is given by (5.6) (Kelley [10]).

$$\|F(\mathbf{x_{k-1}} + \lambda\delta_k)\| < (1 - \alpha\lambda)\|F(\mathbf{x_{k-1}})\| \tag{5.6}$$

and α is a parameter typically set to the value 1×10^{-4}. C.T. Kelley's line search implementation for Broyden's method is a three-point parabolic line search, which is a type of polynomial line search strategy. The idea here is to use the information obtained when the search is rejecting steps. Specifically, one can model the scalar function of (5.7)

$$f(\lambda) = \|F(\mathbf{x_{k-1}} + \lambda\delta_k\|^2 \tag{5.7}$$

with a polynomial and use the minimum of that polynomial as the next step length [10]. In addition to line search methods, there are other popular classes of methods, one example being trust region methods, discussed by Powell in [15].

5.3. Better Jacobian estimates with finite differences. Another problem with algorithms 1 and 2 is that the identity matrix may not be a close enough approximation to the Jacobian in order to obtain reasonable convergence. To avoid this pitfall, BroydenAND uses a finite difference approximation for the initial Jacobian B_0, using the first-order forward difference method with a fixed step size Δx. Algorithm 3 also makes use of finite differences on two other occasions. If the line search reaches the maximum number of iterations, the Jacobian is approximated with forward differences, using the solution from the previous iteration, on line 23. If this finite difference approximation is taken, the Broyden update is skipped.

The other place where forward differences can potentially be used is on line 35 of algorithm 3. This is done if the convergence is poor, in an attempt to improve the convergence for the next iteration. The condition for poor convergence is on line 33; it checks if the absolute value of the difference between the current L_2 error and previous L_2 error, and also the absolute value of the difference between the previous L_2 error and the L_2 error from two iterations back is less than some tolerance. If this is met, the finite difference computation is executed.

6. Results and Comparisons.

6.1. Parameter values and numerical results. The implementations of BroydenAND, using both Broyden's first and second methods, were tested on seven nonlinear vector-valued functions, six of which were taken from the paper by Moré et al. [8], and the seventh from the the paper by Huang et al. [17]. The function definitions can be found in appendix B; the dimension of all seven problems solved by BroydenAND and the competing codes was 100. The results and their respective parameter values are shown in table 6.1. This table includes the number of iterations and function evaluations it took BroydenAND to converge to a tolerance of 1×10^{-6}, where the convergence is measured with the 2-norm of F($\mathbf{x}$). "rest tol" is the tolerance that must be met for the algorithm to determine the convergence is poor enough to warrant estimating the Jacobian with finite differences. The column labeled "max. LS iter" refers to the maximum number of line search iterations allowed before the program reports a line search failure. "ni" and "nfe" refer to the number of iterations through the main loop and the number of function evaluations, repsectively. "rest req." and "lsf" refer to the number of restarts (finite difference approximations of the Jacobian taken on line 35) and the number of line search failures, respectively. A natural question to ask is how the different parameters were found, and why do they have those specific values in the tables. There is unfortunately no profound answer because they are primarily empirical values. The process of running BroydenAND and finding parameters that approximately minimize function evaluations was heuristic; attempts to solve these seven nonlinear functions can be likened to an experiment, rather than a routine deterministic process. The parameters σ_1, σ_2, ρ, and η were all set to the same value for all seven functions. This did not adversely affect the performance, and made it more user-friendly. The values were set to 1×10^{-8}, 1×10^{-8}, $1\text{-}1\times10^{-8}$, and 1×10^{-8}, respectively, so that the condition on line 7 becomes very close to the condition "$if\ \|F(\mathbf{x_{k-1}}+\delta_k)\| < \|F(\mathbf{x_{k-1}})\|\ then...$" and the condition on line 15 becomes very close to the condition "$while\ \|F(\mathbf{x_{k-1}}+\delta_k)\| \geq \|F(\mathbf{x_{k-1}})\|\ do...$". These conditions are more intuitive than simply picking arbitrary values of the parameters. For function 7, some columns have two numbers, separated by a comma. This is because Broyden 1 and 2 did not produce the exact same results for this function; the first number is for Broyden 1 and the second is for Broyden 2.

The SQRF (sequencing QR factorization) and QGN (quasi-Gauss-Newton) methods were written by Wang et al. [18]. These two methods are different implementations of Broyden's first method, and do not use the exact Jacobian, however, it is mentioned on page 61 that the initial Jacobians are approximated with finite differences. Wang and Tewarson only provided the number of iterations, and not function evaluations. Based on the number of iterations it took to converge to a tolerance of 1×10^{-6}, SQRF and QGN do better overall than BroydenAND, with the exception of function 4, where

Func No.	τ	Δx	rest tol	max. LS iter	ni	nfe	rest req.	lsf	SQRF ni	QGN ni	fsolve ni	fsolve nfe	Kinsol ni	Kinsol nfe
1	0.5	1×10^{-2}	1×10^{-2}	10	22	197	0	0	4	4	23	1823	14	1427
2	0.5	1×10^{-2}	1×10^{-3}	10	2	103	0	0	2	2	2	202	2	203
3	0.5	1×10^{-4}	1×10^{-6}	10	25	608	0	4	2	2	197	17697	15	1546
4	0.5	1×10^{-2}	1×10^{-5}	10	8	109	0	0	120	126	6	606	4	405
5	0.5	1×10^{-3}	1×10^{-7}	10	18	119	0	0	9	9	12	1212	12	1213
6	0.5	1×10^{-2}	1×10^{-2}	10	Div	Div	Div	Div	Div	Div	14	1314	Error	Error
7	0.5	1×10^{-2}	1×10^{-6}	10	83,104	1258,1609	0,0	8,10	X	X	47	3747	177	17878

TABLE 6.1
BroydenAND comparisons

BroydenAND performed better. But, of course, the number of iterations is a poor metric for comparison, so the comparisons have very limited value. SQRF, QGN, and BroydenAND all diverged for function 6. The cause is unknown.

BroydenAND was also compared with Matlab's fsolve [6], a function included in the optimization toolbox. The options specified 1st-order forward differencing for the Jacobian derivatives, and it was verified that fsolve uses finite differencing on each iteration. The default solver, trust-region-dogleg, was used, and the solver was asked to converge to a tolerance of 1×10^{-6}. It is assumed that fsolve computes the tolerance with the 2-norm, however, all norms are equivalent and so the results are norm-independent. The superior performance of BroydenAND over fsolve was somewhat expected because fsolve is using finite-differencing on every iteration, whereas BroydenAND only performs finite differences when either the line search fails or the solver stagnates. Fsolve displays the number of function evaluations on-screen; these values were confirmed by using a global variable called "count" inside each function, which is incremented each time the function is called. Fsolve did converge for function 6, which was surprising.

As also shown in table 6.1, the same nonlinear systems were tested with KINSOL [14], which is a member of a suite called SUNDIALS - Suite of Nonlinear and Differential/Algebraic equation Solvers, developed at Lawrence Livermore National Laboratory. KINSOL was specifically written to solve nonlinear algebraic systems. Taking advantage of the Matlab interface, sundialsTB, KINSOL was implemented in Matlab. KINSOL uses Newton's method, and employs forward differencing to approximate the Jacobian on each step. The solver accepts an argument called "strategy", which can be set to either "LineSearch" or "None", and this specifies the global stragegy applied to the Newton step if unsatisfactory. For the first six functions, the strategy was set to "LineSearch", but for function 7, the strategy had to be set to "None" for convergence. The line search algorithm used by KINSOL is unknown. As table 6.1 shows, BroydenAND also out-performs KINSOL in terms of function evaluations. The number of function evaluations used, as reported by KINSOL, was not accurate. As such, the values shown in the tables reflect the values obtained by the aforementioned global count variable. KINSOL was asked to converge to a tolerance of 1×10^{-6}. Interestingly, it was also unable to converge with function 6 regardless of the initial parameters.

The same seven functions were also tested with an implementation of Broyden 1 created by C.T. Kelley, called brsola (Kelley [9]). This algorithm uses Broyden's first method with a three-point parabolic line search. A sufficient step size is determined by the Armijo rule. Brsola did better than BroydenAND on function 1; it required 25 iterations and 79 function evaluations to converge to a tolerance of 1×10^{-6}, while BroydenAND used 197 function evaluations. However, brsola was unable to converge for any of the other six functions. The code was written such that if the line search

fails, the program immediately terminates.

6.2. Graphical results. Table 6.2 shows the results and parameters for convergence of the same seven functions to a tolerance of 1×10^{-10}. In order to achieve reasonable performance on function 5, the step size Δx, to compute the finite differences, had to be decreased to 1×10^{-5}. Figures 6.1 through 6.6 show the convergence history of BroydenAND per function evaluation (corresponding to the results in table 6.2). These results were obtained by putting an fprintf statement in each function to force it to print the 2-norm of F(x) each time it's called. The first 101 function evaluations are omitted from the plots because these are due to initializing the Jacobian approximation, B_0, so the error in this interval is essentially a flat line. Plotted on top of Broyden 1 and Broyden 2 are the fsolve and KINSOL 2-norm errors. Since fsolve and KINSOL take much longer to converge to 1×10^{-10}, they are only plotted until BroydenAND converges. This explains why the fsolve and KINSOL plots are not anywhere near convergence. For the function 2 graph, the fsolve and KINSOL errors lie right on top of each other, and remain approximately constant for the 3 function evaluations shown. The convergence histories of BroydenAND for functions 3 and 7 are highly unsteady; for function 7, the 2-norm of F goes just below 1×10^{8} in some areas. This may suggest that the line search strategy is not performing very well for these functions. The 2-norm of F for Broyden 1 and Broyden 2 is nearly identical for the first five functions, which was expected. However, function 7 is a good example of when Broyden 1 and 2 don't fall right on top of each other. This occurs due to the accumulation of round-off errors. This phenomenon broaches one disadvantage of Broyden's second method, which is its inherent instability. Broyden 1 converges quicker than Broyden 2 in this case, but it was verified that both methods ultimately converge to the same root.

Func No.	τ	Δx	rest tol	max. LS iter	ni	nfe	rest req.	lsf
1	0.5	1×10^{-2}	1×10^{-10}	10	22	197	0	0
2	0.5	1×10^{-2}	1×10^{-10}	10	3	104	0	0
3	0.5	1×10^{-4}	1×10^{-10}	10	33	616	0	4
4	0.5	1×10^{-2}	1×10^{-10}	10	13	114	0	0
5	0.5	1×10^{-5}	1×10^{-10}	10	28	129	0	0
6	0.5	1×10^{-2}	1×10^{-2}	10	Div	Div	Div	Div
7	0.5	1×10^{-2}	1×10^{-10}	10	90,110	1265,1615	0,0	8,10

TABLE 6.2
BroydenAND convergence to 1×10^{-10}

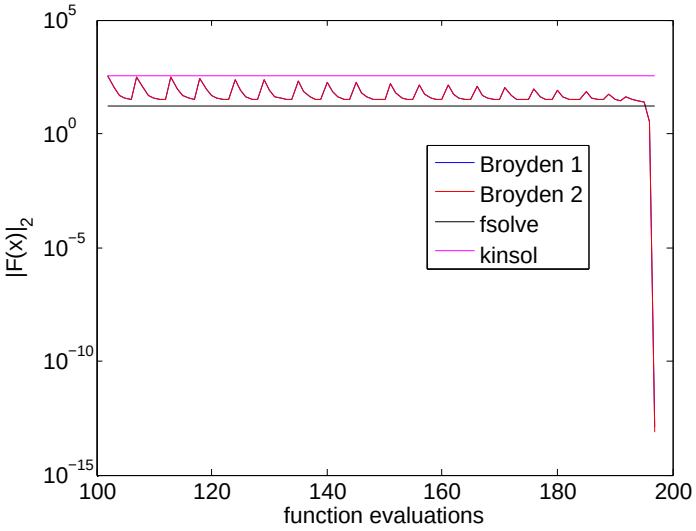

FIG. 6.1. *Function 1 convergence results*

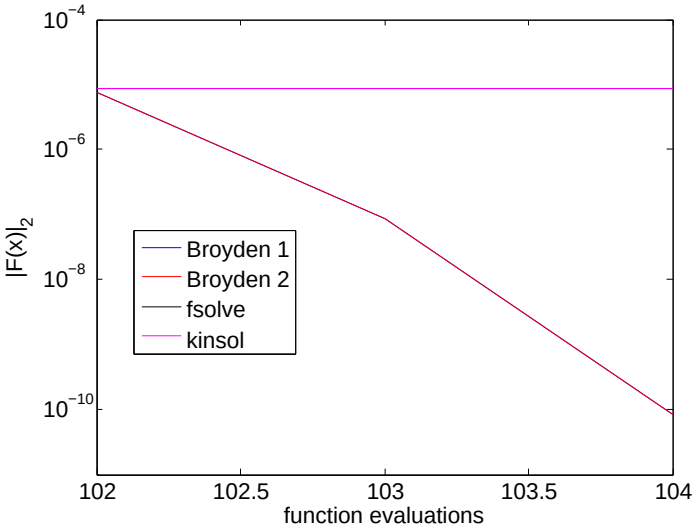

FIG. 6.2. *Function 2 convergence results*

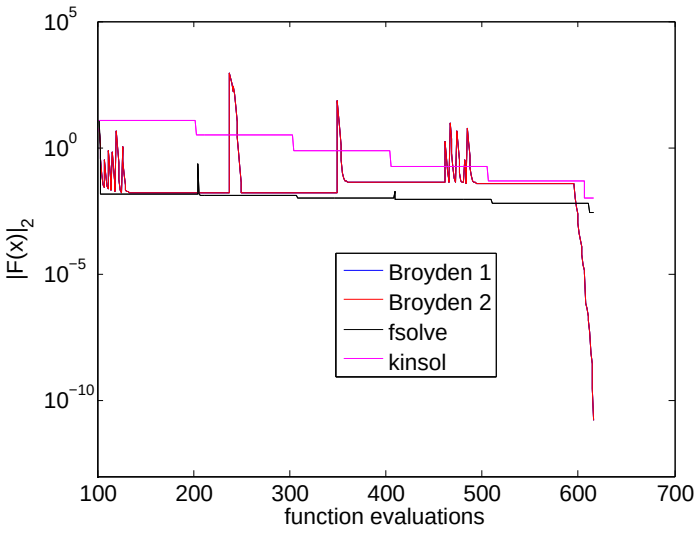

FIG. 6.3. *Function 3 convergence results*

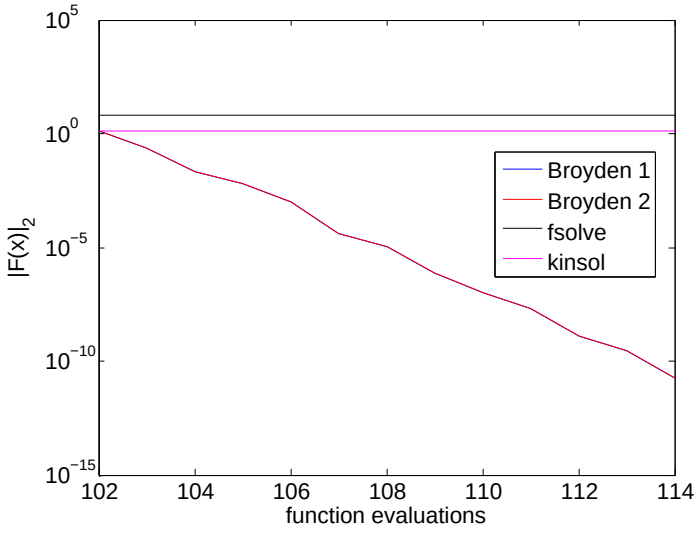

FIG. 6.4. *Function 4 convergence results*

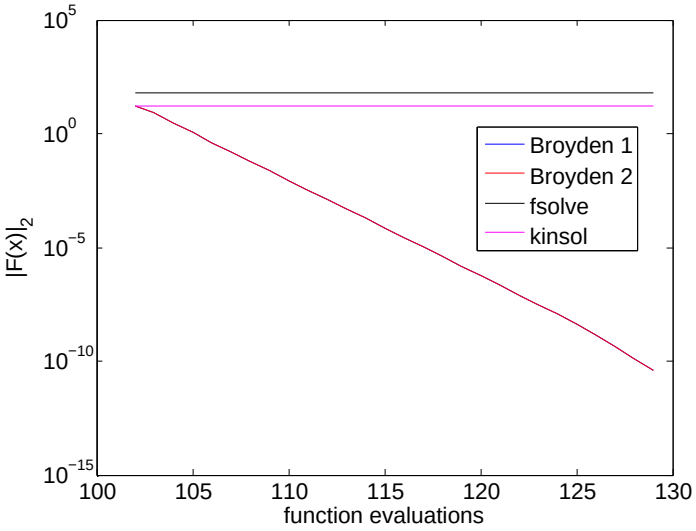

FIG. 6.5. *Function 5 convergence results*

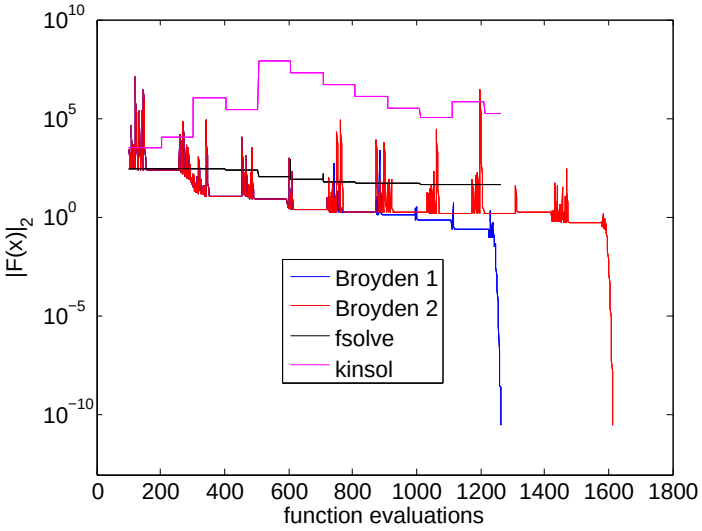

FIG. 6.6. *Function 7 convergence results*

Lastly, it is important to note that for most of the functions, the function evaluation count was mainly dominated by the finite difference computations. With the exception of functions 3 and 7, this was only needed for the initial Jacobian. This may suggest that using a less expensive initial approximation for B would be better (i.e. identity matrix), but this route was tested, and the results were not nearly as promising. Ironically, the finite difference computation was never needed due to lack of convergence, but it was needed due to line search failures for functions 3 and 7.

7. Conclusion. A quasi-Newton algorithm, 'BroydenAND', for solving small (approximately $<$ 100-dimensional) nonlinear systems of equations was proposed. BroydenAND shows reasonable performance on six of the seven nonlinear systems tested. Overall, it out-performs fsolve, KINSOL, and brsola with respect to the number of function evaluations required for convergence. BroydenAND computes the initial Jacobian approximation with 1st-order forward differencing to give the solver a better chance of starting on the right track. This turns out to be paramount in the performance of the solver; simply starting with the identity matrix places more emphasis on fine-tuning other pre-solver knobs, which is undesirable. The solver has the ability to "reset" the Jacobian approximations once it has started, if the line search fails or the convergence has slowed down to an unacceptable rate. Resetting the Jacobian was required for line search failures in functions 3 and 7, but was never required for slow convergence. The convergence history for function 7 clearly shows the divergence of Broyden 2 from Broyden 1, which must be due to round-off error accumulation.

Regarding future work to improve BroydenAND, an improvement of the line search strategy could be useful. In particular, a deeper look into the work by C.T. Kelley could prove to be very helpful in finding a more robust method for step size control. Perhaps a combination of the polynomial line search method with finite difference approximations, which Kelley does mention in [10], would be better than the line search proposed by Li et al. [12].

It is important to remember that the main benefit of BroydenAND is its minimization of function evaluations; and this becomes more important as the functions become more complicated. In extreme cases, the price to compute the Jacobian exactly (or to compute finite differences on every iteration if Newton's method is used) can be exorbitant. It is for these cases that Broyden's methods pay dividends.

8. Appendices.

8.1. Broyden's First Method with Line Search.

Algorithm 3 Broyden 1 with line search

$B_0 = finite\ difference\ approximation$
for k=1,2,3,... **do**
  $\lambda = 1.0$
  $cancel\_update = 0$
  $i = 0$
  $\delta_k = -B_{k-1} \backslash F(\mathbf{x_{k-1}})$
  if $\|F(\mathbf{x_{k-1}} + \delta_k)\| < \rho\|F(\mathbf{x_{k-1}})\| - \sigma_2\|\delta_k\|^2$ **then**
    $\mathbf{x_k} = \mathbf{x_{k-1}} + \delta_k$
    $\mathbf{s_k} = \mathbf{x_k} - \mathbf{x_{k-1}}$
    $\Delta_k = F(\mathbf{x_k}) - F(\mathbf{x_{k-1}})$
    $B_k = B_{k-1} + \frac{(\Delta_k - B_{k-1}\mathbf{s_k})\mathbf{s_k}^T}{\mathbf{s_k}^T\mathbf{s_k}}$
  else
    $\lambda = \lambda\tau$
    $fnorm = \|F(\mathbf{x_{k-1}} + \lambda\delta_k)\|$
    while $fnorm \geq \|F(\mathbf{x_{k-1}})\| - \sigma_1\|\lambda\delta_k\|^2 + \eta^k\|F(\mathbf{x_{k-1}})\|)$ && $i < imax$ **do**
      $\lambda = \lambda\tau$
      $F(\mathbf{x_k}) = F(\mathbf{x_{k-1}} + \lambda\delta_k)$
      $fnorm = \|F(\mathbf{x_k})\|$
      $i = i + 1$
    end while
    if $i == imax$ **then**
      $sprintf('Line\ search\ failure')$
      $compute\ finite\ difference\ approx.\ with\ xold$
      $cancel\_update = 1$
    end if
    $\mathbf{x_k} = \mathbf{x_{k-1}} + \lambda\delta_k$
    $s_k = \mathbf{x_k} - \mathbf{x_{k-1}}$
    $\Delta_k = F(\mathbf{x_k}) - F(\mathbf{x_{k-1}})$
    if $cancel\_update \neq 1$ **then**
      $B_k = B_{k-1} + \frac{(\Delta_k - B_{k-1}\mathbf{s_k})\mathbf{s_k}^T}{\mathbf{s_k}^T\mathbf{s_k}}$
    end if
  end if
  if $abs(\|F(\mathbf{x_k})\| - \|F(\mathbf{x_{k-1}})\|) < restart\ tol$ && $abs(\|F(\mathbf{x_{k-1}})\| - \|F(\mathbf{x_{k-2}})\|) < restart\ tol$ && $cancel\_update \neq 1$ **then**
    $sprintf('Not\ converging\ well\ on\ iteration\ k')$
    $compute\ finite\ difference\ approx.\ with\ xold$
  end if
  if $\|F(\mathbf{x_k})\| < tol$ **then**
    $break$
  end if
  $\mathbf{x_{k-1}} = \mathbf{x_k}$
  $F(\mathbf{x_{k-1}}) = F(\mathbf{x_k})$
end for

8.2. Function Definitions. Function 1: Extended Rosenbrock Function

$n = dimension,\ \ must\ be\ even$
$f_{2i-1}(x) = 10(x_{2i} - x_{i-1}^2)$
$f_{2i}(x) = 1 - x_{2i-1}$
$i = 1, ..., \frac{n}{2}$
$x_0 = (\zeta)\ \ where\ \ \zeta_{2j-1} = -1.2,\ \zeta_{2j} = 1$
$j = 1, ..., \frac{n}{2}$
$f = 0\ at\ (1, ..., 1)$

Function 2: Discrete Boundary Value Function

$n = dimension$
$f_i(x) = 2x_i - x_{i-1} - x_{i+1} + \frac{h^2}{2}(x_i + t_i + 1)^3$
$where\ \ h = \frac{1}{n+1},\quad t_i = ih,\quad x_0 = x_{n+1} = 0$
$i = 1, ..., n$
$x_0 = (\zeta_j)\ \ where\ \ \zeta_j = t_j(t_j - 1)$
$j = 1, ..., n$
$f = 0$

Function 3: Trigonometric Function

$n = dimension$
$f_i(x) = n - \sum_{j=1}^{n} cos(x_j) + i(1 - cos(x_i)) - sin(x_i)$
$i = 1, ..., n$
$x_0 = (\frac{1}{n}, ..., \frac{1}{n})$
$f = 0$

Function 4: Broyden Tridiagonal Function

$n = dimension$
$f_i(x) = (3 - 2x_i)x_i - x_{i-1} - 2x_{i+1} + 1,\ \ where\ \ x_0 = x_{n+1} = 0$
$i = 1, ..., n$
$x_0 = (-1, ... - 1)$
$f = 0$

Function 5: Extended Powell Singular Function

$n = dimension$
$f_{4i-3}(x) = x_{4i-3} + 10x_{4i-2}$
$f_{4i-2}(x) = \sqrt{5}(x_{4i-1} - x_{4i})$
$f_{4i-1}(x) = (x_{4i-2} - 2x_{4i-1})^2$
$f_{4i}(x) = \sqrt{10}(x_{4i-3} - x_{4i})^2$
$i = 1, ..., \frac{n}{4}$
$x_0 = (\zeta_j)\ \ where\ \ \zeta_{4j-3} = 3,\ \zeta_{4j-2} = -1,\ \zeta_{4j-1} = 0,\ \zeta_{4j} = 1$
$j = 1, ..., \frac{n}{4}$
$f = 0\ at\ the\ origin$

Function 6: Brown Almost-linear Function

$n = dimension$
$f_i(x) = x_i + \sum_{j=1}^{n} x_j - (n+1),\ \ 1 \le i < n$
$f_n(x) = (\prod_{j=1}^{n} x_j) - 1$

$x_0 = (\frac{1}{2}, ...\frac{1}{2})$
$f = 0 \; at \; (1, ..., 1, 1^{1-n})$

Function 7: function 17 in [17]
$n = dimension$
$f_i(x) = 3x_i + (x_{i+1} - 2x_i + x_{i-1}) + (x_{i+1} - x_{i-1})^2/4$
$i = 1, ..., n$
$x_0 = 0, \quad x_{n+1} = 20$
$Initial \;\; point \;\; x_0 = (10, ..., 10)^T$

REFERENCES

[1] Charles G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of computation (American mathematical society)*, 19:577593, 1965.

[2] Chun Yuan Deng. A generalization of the sherman-morrison-woodbury formula. *Applied Mathematics Letters*, 24:15611564, 2011.

[3] Peter Deuflhard. A short history of Newton's method. *Documenta Mathematica*, Optimization stories:25–30, 2012.

[4] Andreas Griewank. The "global" convergence of Broyden-like methods with a suitable line search. *J. Austral. Math. Soc. Ser. B*, 28:75–92, 1986.

[5] Andreas Griewank. Broyden updating, the good and the bad! *Documenta Mathematica*, Optimization stories:301–315, 2012.

[6] The MathWorks Inc. Matlab R2013b. http://www.mathworks.com/help/optim/ug/fsolve.html.

[7] Jr. J.E. Dennis and Jorge J. Moré. A characterization of superlinear convergence and its application to quasi-Newton methods. *Mathematics of computation*, 28:549–560, 1974.

[8] Burton S. Garbow Jorge, J. Moré and Kenneth E. Hillstrom. Testing unconstrained optimization software. *Trans. Math. Software*, 7:136–140, 1981.

[9] C. T. Kelley, April 2003. http://www4.ncsu.edu/ ctk/newton/SOLVERS/brsola.m.

[10] C.T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1995.

[11] K. Ming Leung, February 2003. http://cis.poly.edu/ mleung/CS3734/s03/ch02/ShermanMorrison4-out.pdf.

[12] Dong-Hui Li and Masao Fukushima. A derivative-free line search and global convergence of Broyden-like methods for nonlinear equations. *Optimization Methods Software*, 13:181–201, 2000.

[13] José Mario Martínez. Practical quasi-newton methods for solving nonlinear systems. *Journal of Computational and Applied Mathematics*, 124:97–121, 2000.

[14] The Regents of the University of California. Sundials v.2.5.0, March 2012. http://computation.llnl.gov/casc/sundials/main.html.

[15] M. J. D. Powell. A hybrid method for nonlinear equations. *Numerical Methods for Nonlinear Algebraic Equations*, pages 87–114, 1970.

[16] Timothy Sauer. *Numerical Analysis*. Addison-Wesley, Boston, MA, 2006.

[17] E. Spedicato and Z. Huang. Numerical experience with Newton-like methods for nonlinear algebraic systems. *Computing*, 58, 1997.

[18] H. Wang and R. P. Tewarson. A quasi-Gauss-Newton method for solving nonlinear algebraic equations. *Computers Math. Applic.*, 25:53–63, 1993.