Mathematical Programming 23 (1982) 20–33
North-Holland Publishing Company

A DISCRETE NEWTON ALGORITHM FOR MINIMIZING A FUNCTION OF MANY VARIABLES

Dianne P. O'LEARY

Computer Science Department and Institute for Physical Science and Technology, University of Maryland, College Park, MD, U.S.A.

Received 4 August 1980
Revised manuscript received 16 February 1981

A Newton-like method is presented for minimizing a function of n variables. It uses only function and gradient values and is a variant of the discrete Newton algorithm. This variant requires fewer operations than the standard method when $n > 39$, and storage is proportional to n rather than n^2.

Key words: Discrete Newton Algorithm, Function Minimization, Conjugate Gradient.

1. Introduction

Many algorithms have been proposed for finding a local minimum of a function $f(x)$ from R^n into R^1 using only function and gradient values. These include discrete Newton methods, quasi-Newton methods, and the conjugate gradient algorithm. For a discussion of these methods, see, for example [1]. There are classes of problems for which each of these methods is the most efficient.

In this paper, we describe a variant of the discrete Newton algorithm which requires fewer operations than the standard method when $n > 39$. If a preconditioned form of the matrix of second derivatives has many multiple or clustered eigenvalues, this can be exploited, and even greater efficiency can be achieved. Storage is of $O(n)$ rather than the $O(n^2)$ of the standard discrete Newton method, and this method requires less storage if $n > 12$. The algorithm uses a preconditioned conjugate gradient method to generate and solve the linear system which determines the Newton step.

In Section 2 we present the algorithm and discuss its convergence rate and properties. Section 3 provides numerical examples. A Fortran code is provided in [2].

2. A discrete Newton algorithm and its properties

In this section we consider the problem

$$\min_{x \in R^n} f(x).$$

We assume that f is twice continuously differentiable, and we denote the gradient of f by $g(x)$ and the Hessian matrix of second derivatives by $G(x)$. Section 2.1 reviews standard discrete Newton methods for this problem. Section 2.2 discusses a tool used in our algorithm, the conjugate gradient method for solving linear systems. The discrete Newton method variant is presented in Section 2.3 and its properties are discussed in Section 2.4.

2.1. The discrete Newton algorithm

In discrete Newton methods, the Hessian matrix $G(x)$ is approximated by differencing the gradient, using the relation

$$G(x)v = \lim_{h \to 0} \frac{g(x + hv) - g(x)}{h}.$$

The standard method is to difference g along the coordinate directions $e_1, \ldots, e_n$ (where e_i is the ith unit vector) in order to generate approximations to the columns of G. Since, for smooth functions, G is symmetric, the approximate Hessian is then often symmetrized by averaging corresponding elements in the upper and lower triangles.

Directions other than the coordinate ones have been proposed for the differencing; see, for example, Brent [3]. If G is known to have a fortuitous zero–nonzero structure, then other directions are more efficient than coordinate directions for calculating the elements. For example, a tridiagonal Hessian of any dimension n can be approximated with three evaluations of the gradient. This idea of tailoring the direction of the finite difference to the structure of the Hessian has been exploited by Curtis et al. [4], and Powell and Toint [5].

Once the differences have been calculated, however, there remains the problem of solving a linear system of equations in order to determine the Newton step direction p satisfying

$$G(x)p = -g(x).$$

In high dimensions this can be an expensive operation, in general of order n^3, unless special techniques are used as in [6].

There is the added complication of insuring that the step direction actually is a descent direction for small step lengths, i.e., that $p^T g(x) < 0$. Safeguarded Newton algorithms do this by modifying $G(x)$ to make it positive definite if it is not already. The most efficient algorithms for this are based on the Cholesky factorization of a symmetric matrix into a lower triangular matrix times its transpose [7, 8]. A survey of some safeguarding techniques is given in [9].

As a further precaution, many algorithms insist that the function value decrease at each iteration. If the Newton step does not produce a sufficient decrease, a linesearch is performed in the Newton direction.

Near a stationary point, the norm of $g(x)$ may be close to zero but $G(x)$ may

be indefinite. In this case a direction can be substituted which satisfies $p^T G(x)p < 0$. Gill and Murray [7] have shown that such a direction can be calculated easily, given the Cholesky factors of the modified Hessian matrix.

The following discrete Newton scheme is typical of many of the better algorithms.

Standard discrete Newton algorithm

Given: x_0, $g(x_0)$, a steplength h for the finite differences, and a convergence tolerance ϵ

For $k = 0, 1, \ldots$

Step 1. Generate an approximate Hessian matrix G_k by taking finite differences in predetermined directions (usually the coordinate unit vectors) and averaging corresponding elements in the upper and lower triangles.

Step 2. Perform a Cholesky factorization of the approximate Hessian G_k into LL^T, modifying it if necessary to make the factored matrix positive definite.

Step 3. If $\|g(x_k)\| < \epsilon$ and G_k is positive definite, halt with the approximate minimizer x_k.

Step 4. If $\|g(x_k)\| < \epsilon$ and G_k is not positive definite, compute p_k to be a direction of negative curvature of G_k. Otherwise compute p_k to satisfy $LL^T p_k = -g(x_k)$.

Step 5. If $f(x_k + p_k)$ is sufficiently less than $f(x_k)$ (as judged by Goldstein's criteria [10] or some other test), then set $x_{k+1} = x_k + p_k$.

Otherwise determine α so that $f(x_k + \alpha p_k)$ gives a sufficient decrease in the function value and set $x_{k+1} = x_k + \alpha p_k$.

Step 6. Evaluate $g(x_{k+1})$.

The algorithm proposed in Section 2.3 combines Steps 1–4, producing a method which in some cases is significantly more efficient. It uses as a tool the conjugate gradient algorithm, which is reviewed in the next section.

2.2. *The preconditioned conjugate gradient algorithm*

The conjugate gradient algorithm for solving linear systems is an iterative method which can be used to solve $G_k p_k = -g(x_k)$. It requires at each step the product of the matrix G_k with a given vector v. Total space required is proportional to n. To simplify notation, in this section we will drop the subscript k on G_k and p_k and denote $g(x_k)$ by g.

The conjugate gradient method was originally phrased to solve positive definite systems of linear equations [11], but this is not sufficient for our purposes. To preserve stability when G is indefinite we need to exploit the relationship of the conjugate gradient algorithm to one of Lanczos [12, 13]. From this viewpoint the algorithm is a method to factor G into the product of an

orthogonal matrix times a tridiagonal times an orthogonal:

$$G = VTV^T,$$
$$V^TV = I,$$

$$T = \begin{bmatrix} \rho_1 & \bar{\beta}_2 & & & & \\ \bar{\beta}_2 & \rho_2 & \bar{\beta}_3 & & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \bar{\beta}_n \\ & & & & \bar{\beta}_n & \rho_n \end{bmatrix}.$$

The matrices V and T are generated (and discarded) column by column. If a full n steps of the algorithm are taken, then the linear system

$$Gp = -g$$

is equivalent to

$$VTV^Tp = -g$$

or

$$Ty = -V^Tg, \quad y = V^Tp.$$

Typically the first column of V is chosen to be in the direction of $-g$ so that

$$-V^Tg = \|g\|e_1.$$

The linear system for y can be solved using a Cholesky decomposition, modifying the matrix T if necessary to make it positive definite. This gives the equivalent problem

$$LL^Ty = \|g\|e_1$$

where L is lower bidiagonal.

As it stands now, the solution cannot be accumulated in an iterative manner without saving all previous vectors, but an idea of Chandra [14] borrowed from a similar situation discussed by Paige and Saunders [15], overcomes this problem. Let C be defined by

$$V = CL^T.$$

Then the columns of C can be computed sequentially, without saving all of L or V, by forward substitution on the system

$$LC^T = V^T.$$

At the same time, p can be accumulated using the relations

$$Lz = \|g\|e_1,$$
$$p = Vy = VL^{-T}z = Cz.$$

Chandra [14] notes that in order to preserve stability when T is indefinite or close to semi-definite, it is necessary to do a block triangular factorization making L block lower triangular, with 1×1 or 2×2 matrices on its main diagonal. This version of the Cholesky factorization is due to Bunch and Parlett [16]. An alternative to this is to use the scheme of Paige and Saunders [15] where T is factored as the product of a lower triangular matrix $\hat{L}$ and a very simple orthogonal matrix Q. Then, in the relations above, $\hat{L}$ would play the role of L and Q would play the role of L^T. The resulting code requires somewhat more operations.

So, given V and T by columns, we can solve the system. Recurrences for the columns are defined by

$$GV = VT.$$

Writing the jth column of this relation we get

$$Gv_j = \bar{\beta}_j v_{j-1} + \rho_j v_j + \bar{\beta}_{j+1} v_{j+1}$$

where v_j is the jth column of V.

Using the orthogonality relations gives (through an induction argument)

$$\rho_j = v_j^T G v_j,$$
$$\bar{\beta}_{j+1} = \|Gv_j - \bar{\beta}_j v_{j-1} - \rho_j v_j\| = v_j^T G v_{j+1}.$$

The definitions of $\bar{\beta}_{j+1}$ are equivalent if G is symmetric and no round-off error occurs. In computational practice, however, they are not equal. To preserve orthonormality it is best to use the first definition in generating the v sequence. To symmetrize the tridiagonal matrix we use the average of the two values as the new off-diagonal element.

In the course of the factorization, if T is found to be indefinite, the factors can be modified as in Gill and Murray's algorithm [7] to produce a positive definite matrix. In terms of the original matrix,

$$G = VLL^TV^T - VDV^T$$

where D is a positive semi-definite diagonal matrix. We will denote the positive definite matrix VLL^TV^T by $\bar{G}$.

To find a direction of negative curvature when G is indefinite and the gradient is small, we use a method directly analogous to that of Gill and Murray. Let s be the index of the maximum element of the diagonal of D. Then we determine p by solving

$$L^Tz = e_s$$

where e_s is the sth unit vector, and setting $p = Vz$. Then

$$\begin{aligned} p^T G p &= p^T V L L^T V^T p - p^T V D V^T p \\ &= e_s^T e_s - z^T D z \\ &= 1 - \sum_{j=1}^{n} z_j^2 d_{jj}. \end{aligned}$$

Now note that $z_{s+1} = \cdots = z_n = 0$, $z_s = 1/l_{ss}$, $d_{jj} \geq 0$, $j = 1, 2, \ldots, n$, and $l_{ss}^2 - d_{ss} = \rho_s - l_{s,s-1}^2 < 0$ if G is indefinite. Therefore,

$$\begin{aligned} p^T G p &= 1 - \frac{d_{ss}}{l_{ss}^2} - \sum_{j=1}^{s-1} z_j^2 d_{jj} \\ &= \frac{l_{ss}^2 - d_{ss}}{l_{ss}^2} - \sum_{j=1}^{s-1} z_j^2 d_{jj} < 0. \end{aligned}$$

Thus p is indeed a direction of negative curvature.

As noted above, when carried a full n steps, conjugate gradients may be regarded as a direct method. It is often more practical, however, to regard it as an iterative method. If we define

$$\mathscr{E}(x) = 1/2(x - x^*)^T \bar{G}(x - x^*),$$

then a convergence bound is given by [17, 18]:

$$\mathscr{E}(x_k) \leq 4\left(\frac{1 - \kappa^{-1/2}}{1 + \kappa^{-1/2}}\right)^{2k} \mathscr{E}(x_0)$$

where x^* is the true solution to the problem $\bar{G}p = -g$ and $\kappa = \lambda_{\max}(\bar{G})/\lambda_{\min}(\bar{G})$ is the condition number of the matrix $\bar{G}$, the ratio of the largest and smallest eigenvalues. Convergence can be accelerated by preconditioning the problem; see, for example, [19, 20]. Consider the equivalent linear system

$$M^{1/2} \bar{G} M^{1/2} w = -M^{1/2} g$$

where $M^{1/2} w = p$ and $M^{1/2}$ is symmetric positive definite. Writing the conjugate gradient algorithm for this problem and then converting back to the original variables gives a recurrence

$$\bar{G} M V = V T \quad \text{where } V^T M V = I.$$

The convergence bound above is still valid, where κ is now the condition number of $M\bar{G}$. In addition if $M\bar{G}$ has only r distinct eigenvalues, the exact solution will be found in r or fewer iterations. The complete algorithm is given below.

A conjugate gradient algorithm

Initialization: Given a convergence tolerance ϵ, a preconditioning matrix M, a lower bound $\omega^{1/2}$ for main diagonal elements of L, and an upper bound Ω for the

off-diagonal elements of L, let

$$v_0 = c_0 = p = 0,$$
$$\beta_1^{(2)} = (g^T M g)^{1/2}, \qquad v_1 = -g/\beta_1^{(2)},$$
$$\delta_1 = 0.$$

For $j = 1, 2, \ldots$

Step 1. Generate the new Lanczos vector.

$$\rho_j = v_j^T MGMv_j,$$
$$\beta_{j+1}^{(1)} v_{j+1} = GMv_j - \rho_j v_j - \beta_j^{(2)} v_{j-1}$$

where $\beta_{j+1}^{(1)}$ is chosen so that $v_{j+1}^T M v_{j+1} = 1$,

$$\beta_{j+1}^{(2)} = v_j^T MGMv_{j+1}.$$

Step 2. Update the factorization, modifying if necessary to make the tridiagonal matrix positive definite.

$$\bar{\beta}_{j+1} = \frac{\beta_{j+1}^{(1)} + \beta_{j+1}^{(2)}}{2}, \quad \gamma_j = \rho_j - \delta_j^2, \quad d_{jj} = 0.$$

If $\gamma_j < \omega$, then $d_{jj} = \max(\omega, |\gamma_j|) - \gamma_j$, $\delta_{j+1} = \bar{\beta}_{j+1}/\sqrt{\gamma_j + d_{jj}}$.
If $|\delta_{j+1}| > \Omega$, then

$$d_{jj} = d_{jj} - (\gamma_j + d_{jj})(1 - \delta_{j+1}^2/\Omega^2) \quad \text{and} \quad \delta_{j+1} = \operatorname{sgn}(\delta_{j+1})\Omega,$$

$\gamma_j = \sqrt{\gamma_j + d_{jj}}$.

Step 3. Update the iterate; compute the next column of C and the residual norm.

$$z_j = -\delta_j z_{j-1}/\gamma_j \quad (\text{with } z_1 = \beta_1/\gamma_1),$$
$$c_j = (Mv_j - \delta_j c_{j-1})/\gamma_j,$$
$$p = p + z_j c_j,$$
$$\|r\| \equiv \|\bar{G}p + g\| = |z_j \bar{\beta}_{j+1}/\gamma_j|.$$

If $\|r\| < \epsilon$, then halt.

Gill and Murray advise that ω be taken as $2^{-t/2}$ on a t bit machine, and Ω as $\max(\max_i |G_{ii}|^{1/2}, \max_{m \neq i} |G_{im}/n|^{1/2})$. Since in our case Ω is not computable, an overestimate should be used; the effect of this is to keep the difference between G and $\bar{G}$ somewhat smaller by allowing more ill-conditioning in the factors.

2.3. *A discrete Newton variant*

We combine the discrete Newton algorithm of Section 2.1 with the conjugate gradient algorithm of Section 2.2 to obtain the following method.

A discrete Newton variant

Given: x_0, $g(x_0)$, a steplength h for the finite differences, and a convergence tolerance ϵ.

For $k = 0, 1, \ldots$

Step 1. Use the preconditioned conjugate gradient algorithm of Section 2.2 to find p_k to solve the linear system

$$\bar{G}_k p_k = -g(x_k)$$

or, if $\|g(x_k)\| < \epsilon$, to find a direction of negative curvature. The inner product of G_k with a vector v of length 1 is approximated by

$$G_k v = \frac{g(x_k + hv) - g(x_k)}{h}.$$

If G_k is found to be indefinite, the matrix T is modified by adding a diagonal matrix to it. The algorithm is terminated when the residual norm $\|\bar{G}_k p_k + g(x_k)\|$ is sufficiently small.

Step 2. If $\|g(x_k)\| < \epsilon$ and G_k was found to be positive definite, then halt with the approximate minimizer x_k.

Step 3. If $f(x_k + p_k)$ is sufficiently less than $f(x_k)$ (as judged by the Goldstein criteria or some other test), then set

$$x_{k+1} = x_k + p_k.$$

Otherwise determine α so that $f(x_k + \alpha p_k)$ gives a sufficient decrease and set

$$x_{k+1} = x_k + \alpha p_k.$$

Step 4. Evaluate $g(x_{k+1})$.

An algorithm related to this one, but applicable only when f is convex and the approximate Hessian is positive definite, was presented in [21, p. 137] as a special case of one in [22].

2.4. Properties of the algorithm

The standard discrete Newton method takes, in general, n gradient evaluations and $\frac{3}{2}n^2$ multiplications and additions to evaluate and symmetrize G_k, and, ignoring any modifications to make G_k positive definite, $\frac{1}{6}n^3 + \frac{3}{2}n^2$ multiplications and additions for the factorization and solution of the linear system. Storage required is $\frac{1}{2}n^2$. Conjugate gradients with no preconditioning, and assuming the worst case where a full n iterations are necessary to solve the linear system to an acceptable level of accuracy, involves n gradient evaluations and n^2 multiplications and additions to form the product of G_k with the direction vectors, plus $9n^2$ multiplications and $7n^2$ additions as overhead. The $\frac{1}{2}n^2$ storage locations of the standard method are replaced by $6n$.

Thus, this variant requires fewer operations when $n > 39$ and less storage when $n > 12$.

Preconditioning the conjugate gradient iteration can reduce the number of gradient evaluations by reducing the number of conjugate gradient iterations necessary to determine the direction. Possible choices of the preconditioning matrix include the following:

(1) M^{-1} can be chosen to be $G(x_0)$, the Hessian matrix at the initial point.

(2) M^{-1} can be chosen as a part of the Hessian that is easy to evaluate, perhaps a constant portion. If, for example, the Hessian is a constant matrix plus a matrix of rank r, conjugate gradients will, in the absence of rounding and truncation error, terminate with the exact solution in at most r iterations when using this preconditioning. Acceleration will also occur if the non-constant portion is small in norm. If storage is not sufficient to store a factorization of the chosen portion of the Hessian, a partial factorization might be used instead.

Information saved from a previous conjugate gradient iteration can also be used to accelerate conjugate gradients. If V_j $(n \times j)$ and T_j $(j \times j)$ are the orthogonal and tridiagonal matrices computed in the first j conjugate gradient steps at Newton iteration k, then the initial guess for p at a later Newton iteration can be defined as $-V_j T_j^{-1} V_j^T g$ rather than 0, with $g + Gp$ substituted for g in the definitions of $\beta_1^{(2)}$ and v_1 in the conjugate gradient algorithm. If the Hessian matrix has not changed much since iteration k, then very few conjugate gradient iterations will be required.

The convergence rate of this discrete Newton algorithm is best understood intuitively by breaking the error into three pieces. Let x^* be the true solution to the problem, $G(x_k)$ the Hessian evaluated at x_k, G_k the approximate Hessian obtained by differencing, and H_k the matrix that conjugate gradients actually used, i.e., $p_k = -H_k g_k$. Then, if the step length chosen was 1 and if $G(x_k)$ is positive definite,

$$\begin{aligned} x_{k+1} - x^* &= x_k - H_k g_k - x^* \\ &= (x_k - G(x_k)^{-1} g_k - x^*) + (G(x_k)^{-1} - G_k^{-1}) g_k + (G_k^{-1} - H_k) g_k. \end{aligned}$$

Therefore,

$$\begin{aligned} \|x_{k+1} - x^*\| \leq \|x_k - G(x_k)^{-1} g_k - x^*\| &+ \|(G(x_k)^{-1} - G_k^{-1}) g_k\| \\ &+ \|(G_k^{-1} - H_k) g_k\|. \end{aligned}$$

The first term is the Newton error at the $(k+1)$st step. The second term is error due to discrete differencing and depends on the choice of h in the conjugate gradient algorithm. The third term is error due to round-off and early termination in conjugate gradients. Strategies for balancing the first two errors have been studied in detail. If, for example, f is sufficiently smooth, G_k is a strongly consistent approximation to $G(x_k)$, and $G(x^*)$ is nonsingular, then local quadratic convergence can be established if the finite difference step length h

decreases sufficiently rapidly as k increases [23]. To obtain the same convergence for the conjugate gradient variant, this implies that the third term in the error should go to zero as fast as the second; thus, the ϵ used as the convergence tolerance for conjugate gradients and the finite difference parameter h must both decrease sufficiently rapidly as k increases. Formal results of this nature can be derived from those of Bus [24].

3. Computational results and conclusions

The discrete Newton variant was programmed and tested on a UNIVAC 1100/40 computer in double precision arithmetic (approx. 18 decimal digits). Performance was compared with a standard discrete Newton algorithm, which differed only in that the Hessian approximation was formed by differencing along the coordinate axes and then symmetrizing, and the linear system was solved using a Cholesky factorization with modification to force the matrix to be positive definite.

Four test problems were used:

(1) Generalized Rosenbrock function [25, 8]

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad \text{if} \quad n = 2,$$

$$f(x) = 1 + \sum_{i=2}^{n} (100(x_i - x_{i-1}^2)^2 + (1 - x_i)^2 \quad \text{if} \quad n > 2.$$

Initial guess:

$$x = (-1.2, 1.0)^T \quad \text{if} \quad n = 2$$

$$x = (1/(n+1), \ldots, n/(n+1))^T \quad \text{for} \quad n = 50, 100.$$

(2) Watson problem [25]

$$f(x) = \sum_{i=1}^{30} \left(\sum_{j=1}^{n} x_j (j-1) \left(\frac{i-1}{29} \right)^{j-2} - \left(\sum_{j=1}^{n} x_j \left(\frac{i-1}{29} \right)^{j-1} \right)^2 - 1 \right)^2 + x_1^2.$$

Initial guess:

$$x = (0, \ldots, 0)^T, \quad n = 6.$$

(3) Powell problem [8]

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4.$$

Initial guess:

$$x = (3, -1, 0, 1)^T, \quad n = 4.$$

(4) Pen 1 [25]

$$f(x) = \sum_{i=1}^{n} (x_i - 1)^2 + 10^{-3} \left(\sum_{i=1}^{n} x_i^2 - 0.25 \right)^2.$$

Initial guess:

$$x = (1/(n+1), \ldots, n/(n+1))^T \quad \text{or} \quad (1, -1, 1, -1, \ldots)^T \quad \text{for} \quad n = 50, 100.$$

The references given refer to comparable results rather than to the original sources for the problems. Numerical results on these problems are, of course, insufficient to draw firm conclusions, but do indicate the method's promise.

The line search was a modified implementation of an algorithm suggested by Osborne [26]. If a steplength guess of 1 passes Goldstein's test, then it is used. Otherwise parameters are successively determined by one of two algorithms: quadratic interpolation on an interval starting at zero, if the last guess was an overestimate; or the secant method on an interval starting with the previous guess, if the last was an underestimate. This is continued until Goldstein's test is passed. This strategy works well on most problems, but occasionally produces a sequence of bad step lengths which are very close together. This happened, for example, on the Rosenbrock functions. To avoid this problem, Osborne's choice of step length parameter was averaged in the proportion (0.9, 0.1) with the result of bisection on the same interval. This is a very crude strategy, and implementing a high quality line search procedure would undoubtably improve the results, especially on problems for which gradient evaluations are expensive.

In the experiments, the finite difference step length was taken as $h = 10^{-8}$. The conjugate gradient algorithm iteration was terminated when the Euclidean norm of the residual had been reduced by a factor of 10^{-5}. The Goldstein parameter σ was taken as 10^{-3}, demanding very little reduction in the function value per Newton iteration, or 0.25, demanding a substantial reduction. The termination criterion was

$$f(x_k) - f(x^*) \leq 10^{-5}(1 + |f(x^*)|).$$

Sample results are given in Table 1, which lists number of iterations, function evaluations, and gradient evaluations. For comparison with [8], to obtain an accuracy of 10^{-20} rather than 10^{-5} in the function value took 24 Newton iterations (33 function evaluations) for Rosenbrock's example and 39 Newton iterations (73 function evaluations) for Powell's. Results in [8], using a modified Newton algorithm are 15 (57) for Rosenbrock and 22 (67) for Powell. Differences are due to our use of a different line search and discrete approximations to the Hessian.

Pen 1 is a very special test problem in that the Hessian matrix at the solution has $n - 1$ clustered eigenvalues of order 1 and one of order 10^{-3}. The conjugate gradient algorithm can exploit this, and thus this Newton variant performs spectacularly well on this example, even better than the nonlinear conjugate gradient algorithms tested in [25].

The generalized Rosenbrock function is a hard problem for two reasons: the Hessian has no eigenvalue clustering and there are two minimizers for the problem, one with $x_1 = 1$ and one with $x_1 = -1$. This is the only problem for which indefinite Hessians were encountered in the new variant or in which the

Table 1
Numerical results: New variant (standard discrete Newton)

Problem	σ	Number of Newton iterations	Number of function evaluations	Number of gradient evaluations	Number of CG iterations
Rosenbrock $n = 2$	10^{-3}	22 (22)	31 (31)	67 (67)	44 (0)
Watson $n = 6$	10^{-3}	24 (10)	25 (11)	193 (71)	144 (0)
Powell $n = 4$	10^{-3}	11 (11)	12 (12)	56 (56)	44 (0)
Pen 1, $n = 50$ First guess	10^{-3}	2 (2)	3 (5)	7 (203)	4 (0)
Pen 1, $n = 50$ Second guess	10^{-3}	3 (4)	4 (5)	10 (205)	6 (0)
Pen 1, $n = 100$ First guess	10^{-3}	3 (3)	4 (7)	10 (304)	6 (0)
Pen 1, $n = 100$ Second guess	10^{-3}	3 (3)	4 (4)	10 (304)	6 (0)
Gen. Rosenbrock $n = 50$	10^{-3}	38 (64)	106 (203)	1551 (3265)	1474 (0)
	0.25	35 (72)	129 (294)	1373 (3673)	1302 (0)
Gen. Rosenbrock $n = 100$	10^{-3}	88 (*)	268	3994	3817
	0.25	63 (*)	258	2616	2489

(*) No data due to expense.

secant algorithm was employed. The number of iterations was less than that reported for quasi-Newton algorithms in [25], but the number of gradient evaluations is substantial.

Comparison with the standard discrete Newton algorithm shows generally a comparable number of Newton iterations, with variations (up to a factor of 2) on difficult problems due to discretization and round-off error of comparable size but different direction.

The algorithm is relatively insensitive to the choice of parameters. Reducing the finite difference step length to 10^{-11} did not have much effect except on Watson's function, where the work was approximately doubled. Changing the Goldstein parameter also had little effect; results for all functions except the Rosenbrock examples were identical.

Recently, several other algorithms have been independently proposed which exploit the conjugate gradient algorithm in a similar way but do not find a discrete Newton direction, and sometimes lack numerical stability. These methods include [27, 28, 29]. Combining the best features of these methods and the one developed in this paper would probably produce an even more effective algorithm.

In conclusion, this discrete Newton algorithm has advantages in storage and

operations counts over other discrete Newton methods for moderate and large scale problems without special sparsity structure. It retains the standard advantages which discrete Newton algorithms have over quasi-Newton and conjugate gradient algorithms: it is more robust on difficult problems, and it is relatively easy to check necessary and sufficient conditions for the solution. Techniques for preconditioning linear systems can be used to reduce the number of gradient evaluations and on certain classes of problems this will produce a dramatic increase in efficiency. If linear constraints are present in the problem, the techniques described here could be used to compute a projected Newton direction.

Acknowledgment

I wish to thank Charles (Jerry) Huller for helpful discussions concerning matrix modification techniques. The referees made several helpful comments. This work was supported by the Office of Naval Research under Grant N00014-76-C-0391.

References

[1] W. Murray, *Numerical methods for unconstrained optimization* (Academic Press, New York, 1972).

[2] D.P. O'Leary, "A discrete Newton algorithm for minimizing a function of many variables", Computer Science Department Report TR-910, University of Maryland (June 1980).

[3] R.P. Brent, "Some efficient algorithms for solving systems of nonlinear equations", *SIAM Journal on Numerical Analysis* 10 (1973) 327–344.

[4] A.R. Curtis, M.J.D. Powell and J.K. Reid, "On the estimation of sparse Jacobian matrices", *Journal of the Institute of Mathematics and its Applications* 13 (1974) 117–119.

[5] M.J.D. Powell and Ph.L. Toint, "On the estimation of sparse Hessian matrices", *SIAM Journal on Numerical Analysis* 16 (1979) 1060–1074.

[6] M.C. Bartholomew-Biggs, "A matrix modification method for calculating approximate solutions to systems of linear equations", *Journal of the Institute of Mathematics and its Applications* 23 (1979) 131–137.

[7] P.E. Gill and W. Murray, "Newton-type methods for unconstrained and linearly constrained optimization", *Mathematical Programming* 7 (1974) 311–350.

[8] S. Kaniel and A. Dax, "A modified Newton's method for unconstrained minimization", *SIAM Journal on Numerical Analysis* 16 (1979) 324–331.

[9] P.E. Gill and W. Murray, "Newton-type methods for linearly constrained optimization", in: P.E. Gill and W. Murray, eds., *Numerical methods for constrained optimization* (Academic Press, New York, 1974) pp. 29–66.

[10] A.A. Goldstein, "On steepest descent", *SIAM Journal on Control* 3 (1965) 147–151.

[11] M.R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems", *Journal of Research of the National Bureau of Standards* 49 (1952) 409–436.

[12] C. Lanczos, "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators", *Journal of Research of the National Bureau of Standards* 45 (1950) 255–282.

[13] C. Lanczos, "Solution of systems of linear equations by minimized iterations", *Journal of Research of the National Bureau of Standards* 49 (1952) 33–53.
[14] Rati Chandra, "Conjugate gradient methods for partial differential equations", Ph.D. Thesis, Report 129, Department of Computer Science, Yale University (1978).
[15] C.C. Paige and M.A. Saunders, "Solutions of sparse indefinite systems of linear equations", *SIAM Journal on Numerical Analysis* 12 (1975) 617–629.
[16] J.R. Bunch and B.N. Parlett, "Direct methods for solving symmetric indefinite systems of linear equations", *SIAM Journal on Numerical Analysis* 8 (1971) 639–655.
[17] J.W. Daniel, "The conjugate gradient method for linear and nonlinear operator equations", *SIAM Journal on Numerical Analysis* 4 (1967) 10–26.
[18] S. Kaniel, "Estimates for some computational techniques in linear algebra", *Mathematics of Computation* 20 (1966) 369–378.
[19] O. Axelsson, "Solution of linear systems of equations: Iterative methods", in: V.A. Barker, ed., *Sparse matrix techniques* (Springer-Verlag, New York, 1977).
[20] P. Concus, G.H. Golub and D.P. O'Leary, "A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations", in: J.R. Bunch and D.J. Rose, eds., *Sparse matrix computations* (Academic Press, New York, 1976) pp. 309–332.
[21] M.R. Hestenes, *Conjugate direction methods in optimization* (Springer-Verlag, New York, 1980).
[22] R.F. Dennemeyer and E.H. Mookini, "CGS algorithms for unconstrained minimization of functions", *Journal of Optimization Theory and Applications* 16 (1975) 67–85.
[23] J.M. Ortega and W.C. Rheinboldt, *Iterative solution of nonlinear equations in several variables* (Academic Press, New York, 1970).
[24] J.C.P. Bus, "Convergence of Newton-like methods for solving systems of nonlinear equations", *Numerische Mathematik* 27 (1977) 271–281.
[25] P.E. Gill and W. Murray, "Conjugate gradient methods for large-scale nonlinear optimization", Systems Optimization Lab Report SOL-79-15, Department of Operations Research, Stanford University (1979).
[26] M.R. Osborne, "An efficient weak line search with guaranteed termination", Report 1870, Mathematics Research Center, University of Wisconsin (1978).
[27] N.K. Garg and R.A. Tapia, "QDN: A variable storage algorithm for unconstrained optimization", Department of Mathematical Sciences Report, Rice University (1980).
[28] R.S. Dembo and T. Steihaug, "Truncated-Newton algorithms for large-scale unconstrained optimization", School of Organization and Management, Yale University Preliminary Draft (September 1980).
[29] P.E. Gill, W. Murray and S.G. Nash, "Newton-type minimization using the linear conjugate gradient method", Draft, Department of Operations Research, Stanford University (October 1980).