SIAM J. OPTIM.
Vol. 10, No. 1, pp. 56–69

REDUCED STORAGE, QUASI-NEWTON TRUST REGION APPROACHES TO FUNCTION OPTIMIZATION*

LINDA KAUFMAN†

Abstract. In this paper we consider several algorithms for reducing the storage when using a quasi-Newton method in a dogleg–trust region setting for minimizing functions of many variables. Secant methods require $O(n^2)$ locations to store an approximate Hessian and $O(n^2)$ operations per iteration when minimizing a function of n variables. This storage requirement becomes impractical when n becomes large. Our algorithms use a BFGS update and require kn storage and $4kn + O(k^2)$ operations per iteration, but they may require more iterations than the standard trust region techniques. Typically k is between 10 and 100. Our dogleg–trust region strategies involve expressions with matrix products with both the inverse of this Hessian and with the Hessian itself. Our techniques for updating expressions for the Hessian and its inverse can be used to improve the performance of line search, limited memory algorithms.

Key words. quasi-Newton, trust region, limited memory

AMS subject classifications. 49M10, 65K10, 15A23

PII. S1052623496303779

1. Introduction.

Quasi-Newton methods are iterative methods for minimizing a function $f(x)$, where $x \in R^n$, using only first derivative and function information. If $f(x)$ is a quadratic function, quasi-Newton methods converge in at most n iterations in exact arithmetic with exact line searches. At each iteration a linear system is solved using a matrix B which is an approximation to the true Hessian G, which has components $g_{i,j} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}$. At each iteration the matrix B is updated to the matrix B_+ by a rank 1 or 2 update so that at the next iterate x^+, B_+ satisfies the quasi-Newton condition

$$B_+ s = y, \tag{1.1}$$

where $s = x^+ - x$ and $y = \nabla f(x^+) - \nabla f(x)$.

Quasi-Newton methods with reduced storage have been discussed by Nocedal [14], Buckley and LeNir [2], Liu and Nocedal [11], and Byrd, Nocedal, and Schnabel [4]. Their main idea is that by saving the s's and y's of the last k iterations and some of their inner products, one can easily generate a quasi-Newton search direction that is based on the previous k iterations. Thus one would choose a value of k based on the amount of storage or the expense of each iteration for the particular application, and for the first k iterations, the iterates from the reduced storage scheme and the quasi-Newton method would coincide, but on iteration $k + 1$, information from the first iteration would be discarded and the approximation would not necessarily agree with the $(k + 1)$st step of the traditional quasi-Newton approach.

Dogleg–trust region approaches, as in Dennis, Gay, and Welsch [6] and Dennis and Mei [7] for minimizing a function $f(x)$, determine a radius of trust τ which defines the region where one trusts the second-order model of the function to be minimized. The next iterate x^+ must satisfy

$$||x^+ - x||_2 \leq \tau. \tag{1.2}$$

*Received by the editors May 17, 1996; accepted for publication (in revised form) September 10, 1998; published electronically October 20, 1999.
http://www.siam.org/journals/siopt/10-1/30377.html
†Bell Laboratories, Room 2C-461, 700 Mountain Ave., Murray Hill, NJ 07974 (lck@lucent.com).

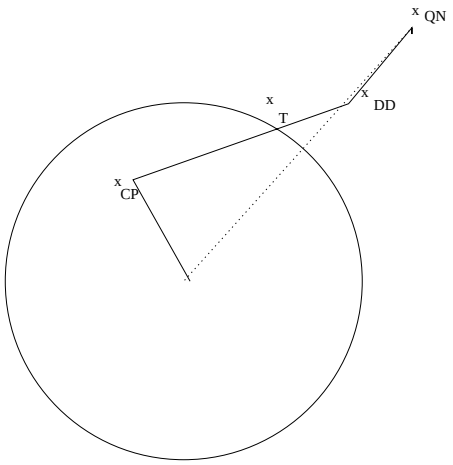

FIG. 1.1. *The double-dogleg path.*

In most trust region approaches, the person who wishes to minimize $f(x)$ initially chooses τ, and its value is gradually changed by the trust region algorithm as the method proceeds.

At each iteration, dogleg–trust region methods consider $\phi(d)$, a quadratic model of $f(x+d)$, of the form

$$\phi(d) = \frac{1}{2} d^T B d + \nabla f^T d + f(x). \tag{1.3}$$

Here B is an approximate Hessian that has been updated according to condition (1.1). If the "quasi-Newton" point $x_{QN} = x - B^{-1}\nabla(fx)$ satisfies (1.2) it becomes a trial step; otherwise the trial step x_T in the double-dogleg strategy is taken as the largest step that satisfies (1.2) and lies on the polygonal line that runs from x to the Cauchy point, x_{CP} (the minimum of (1.3) along the steepest descent direction), to a point x_{DD} in the quasi-Newton direction, up to the x_{QN}, as shown in Figure 1.1. The point x_{DD} is chosen as in Dennis and Mei [7] so that ϕ is guaranteed to monotonically decrease along the polygonal line. If $f(x_T) > f(x)$, τ is decreased and the process is repeated.

The calculation of the dogleg–trust region step involves considering both the quasi-Newton direction and the steepest descent direction at each iteration. Determining the quasi-Newton direction requires the solution of a linear system with B. Adjusting the radius of trust and determining whether the Cauchy point along the steepest descent direction is within that region requires expressions with B.

For more than a decade, the quantum chemistry group at Bell Laboratories has been successfully using the double-dogleg strategy of Dennis and Mei [7], as implemented in MINOP, an early trust region code, which had been changed by the current author to work with the LDL^T decomposition of B updated according to the BFGS formula. The chemists had been drawn to this type of algorithm because their models could be trusted only locally; this paper was prompted by a chemist's reduced storage request.

In section 2 of this paper we construct an algorithm that combines the limited storage algorithms of Byrd, Nocedal, and Schnabel [4] with a dogleg–trust region approach in which a step s can be written as $s = \alpha\eta + \beta\nabla f(x)$, where η is the quasi-

Newton step and α and β are determined so that the step lies within the region of trust. Our algorithm requires at most $4nk + O(k^2)$ multiplications in overhead per iteration. The highest order term is the same as the highest order term required by a line search technique based on [4] that updates B^{-1}. If an algorithm requires B, the schemes given in [4] require $4nk + k^3/6 + O(k^2)$ multiplications. The updating techniques given in this paper can be combined with a limited memory line search approach to yield an algorithm that does not require $O(k^3)$ operations per iteration even when B is required. In section 3 we present some computational evidence that indicates that the limited memory dogleg approach is a viable alternative.

Burke and Weigmann [3] have recently proposed an algorithm for limited memory based on the trust region framework given in Moré and Sorensen [12], which does not use a double-dogleg step but is based on solving $(B + \mu I)d = \nabla f(x)$ for a prescribed value of μ. Their updating scheme also is based on [4] and sometimes has a $k^3/6$ component in the operation count for an iteration.

2. Algorithms. In dogleg–trust region methods the matrix B, the approximate quasi-Newton Hessian, and its inverse appear in several contexts. Given the current iterate x, its function value $f(x)$, the ∇g at x, and the current approximation of the Hessian B, each iteration of the algorithm of Dennis and Mei [7] as explained in Dennis and Schnabel [8] is essentially as follows.

1. Compute the quasi-Newton direction $\eta = -B^{-1}g$.
2. If the quasi-Newton step is within the trust region, i.e., if $||\eta||_2 \leq \tau$, then
   set $s = \eta$,

else
   if the trust region includes a point between DD and
   QN in Figure 1.1, i.e., if $||t\eta|| \leq \tau$ where $c = ||g||^4/((g^T B^{-1} g)(g^T B g))$
   and $t = 0.2 + 0.8c$, then
      set $s = \tau/||\eta||\eta$,
   else
      if the Cauchy step, $p = -(||g||/g^T B g)g$, is outside the radius
      of trust, i.e., $||p|| \geq \tau$, then
         set $s = -(\tau/||g||)g$
      else
         set $s = p + \theta w$, where $w = t\eta - p$,
         $\theta = \sigma/(\phi + (\phi^2 + ||w||^2\sigma)^{1/2})$, $\sigma = \tau^2 - ||p||^2$, and $\phi = p^T w$.

3. Set the new x, $x^+ = x + s$. If $f(x^+) > f(x)$,
   set $\tau = \tau/2$ and go to step 2.
4. The radius τ can be updated as follows:
The predicted function difference, df_m, is $g^T s + 0.5 s^T B s$.
If $(f(x) - f(x^+)) > 0.1 df_m$, then
   set $\tau = 0.5||s||$ and go to step 5
else
   if $|df_m - (f(x) - f(x^+))| \leq 0.1(f(x) - f(x^+))$ or
   $(f(x) - f(x^+)) < .75 df_m$ or $g^T s \geq 2x^T \nabla f(x^+)$, then
      set $\tau = 2||s||$,
   else
      set $\tau = ||s||$.
5. Update the B matrix.

From step 1 on the trust region double-dogleg algorithm we see that we need to compute

$$\eta = B^{-1}\nabla f(x). \tag{2.1}$$

From step 2 we might need

$$\nabla f(x)^T B \nabla f(x). \tag{2.2}$$

From step 4 we need to compute

$$s^T B s. \tag{2.3}$$

The fact that one often can write

$$s = \alpha\eta + \beta\nabla f(x) \tag{2.4}$$

means that $Bs = -\alpha\nabla f(x) + \beta B\nabla f(x)$, so that the product Bs implied in (2.3) really does not have to be computed as long as $B\nabla f(x)$ itself is available or β is zero. In [6], $s = \alpha\eta + \beta D\nabla f(x)$ for some diagonal matrix D, but (2.2) is changed to $\nabla f(x)^T DBD\nabla f(x)$. Thus it appears that at each iteration one needs a representation of B^{-1} that can be inserted into (2.1), and if the quasi-Newton step is outside the trust radius, one needs a representation of B to insert into (2.2).

The ease with which the quantities in (2.1)–(2.3) are computed depends on one's representation of B. In [4] Byrd, Nocedal, and Schnabel suggest the following compact representation based on the BFGS update scheme for the approximate Hessian.

Let x_k represent the kth quasi-Newton iterate, $g(x_k) = \nabla f(x_k)$, $s_k = x_k - x_{k-1}$, and $y_k = g_k - g_{k-1}$. Let $p = \max(1, k - \hat{k})$, where $\hat{k}$ represents the number of iterates for which one has sufficient storage, and let $k' = \min(\hat{k}, k)$. Let

$$S_k = [s_p, \ldots, s_k], \qquad Y_k = [y_p, \ldots, y_k].$$

Let E_k be the $k' \times k'$ diagonal matrix

$$E_k = \operatorname{diag}[s_p^T y_p, \ldots, s_k^T y_k]$$

and let $Z^{(k)}$ be the $k' \times k'$ lower triangular matrix

$$z_{i,i'}^{(k)} = \begin{cases} s_{i+p-1}^T y_{i'+p-1} & \text{if } i > i', \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

Let C_k be the $2k' \times 2k'$ symmetric indefinite matrix

$$C_k = \begin{bmatrix} -E_k & Z^{(k)^T} \\ Z^{(k)} & \delta S_k^T S_k \end{bmatrix}. \tag{2.6}$$

Then, as Byrd, Nocedal, and Schnabel show in [4], the BFGS approximation B_k to the Hessian matrix can be written as

$$B_k = \delta I - [Y_k : \delta S_k]\, C_k^{-1} \begin{bmatrix} Y_k^T \\ \delta S_k^T \end{bmatrix}. \tag{2.7}$$

If we let

$$A_k = [Y_k : S_k]\,,$$

then we may write (2.7) as

$$B_k = \delta I - A_k W_k A_k^T, \tag{2.8}$$

where

$$W_k = \begin{bmatrix} I & 0 \\ 0 & \delta \end{bmatrix} C_k^{-1} \begin{bmatrix} I & 0 \\ 0 & \delta \end{bmatrix}.$$

As shown in [4], one can express C_k^{-1} as

$$C_k^{-1} = \begin{bmatrix} -E_k^{1/2} & E_k^{-1/2} Z^{(k)^T} \\ 0 & J_k^T \end{bmatrix}^{-1} \begin{bmatrix} E_k^{1/2} & 0 \\ -Z^{(k)} E_k^{-1/2} & J_k \end{bmatrix}^{-1}, \tag{2.9}$$

where J_k is a *lower* triangular matrix satisfying

$$J_k J_k^T = V_k = \delta S_k^T S_k + Z^{(k)} E_k^{-1} Z^{(k)^T}. \tag{2.10}$$

Because we are implementing a quasi-Newton method, we may assume that $s_k = B_k y_k$, which implies that $y_k^T s_k = y_k^T B_k y_k > 0$ since B is positive definite. Thus E is positive definite, and $E^{1/2}$ exists and can be written as a positive definite diagonal matrix. Also, V_k in (2.10) must be positive definite and J_k must exist.

Byrd, Nocedal, and Schnabel [4] show that for the BFGS update in (2.7), the matrix $B_k^{-1} = H_k$ can be expressed as

$$H_k = \delta^{-1} I + A_k \begin{bmatrix} 0 & \delta^{-1} I \\ T^{(k)-T} & 0 \end{bmatrix} \begin{bmatrix} (E_k + \delta^{-1} Y_k^T Y_k) & -I \\ -I & 0 \end{bmatrix} \begin{bmatrix} 0 & T^{(k)^{-1}} \\ \delta^{-1} I & 0 \end{bmatrix} A_k^T, \tag{2.11}$$

where

$$t_{i,i}^{(k)} = \begin{cases} s_{i+p-1}^T y_{i'+p-1} & \text{if } i \le i', \\ 0 & \text{otherwise.} \end{cases} \tag{2.12}$$

Since the matrix T is upper triangular, it is easy to apply T^{-1} to a vector by simply solving the appropriate upper triangular system.

Thus determining H_k does not require refactoring any matrix and is quite straightforward.

Given the representation in (2.11) one can compute $\eta = -H_k \nabla f(x_k)$ in (2.1) as follows.

ALGORITHM QN.

1. Determine $u_k = A_k^T \nabla f(x_k)$.
2. Partition u as $\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \begin{matrix} k' \\ k' \end{matrix}$.
3. Solve $T^{(k)} w = u_2$
4. Set $m = (E_k + \delta^{-1} Y_k^T Y_k) w - \delta^{-1} u_1$.
5. Solve $T^{(k)^T} p = m$.
6. Set $\eta = -\delta^{-1} g_k - A_k \begin{bmatrix} -\delta^{-1} w \\ p \end{bmatrix}$.

Because E_k is diagonal and $T^{(k)}$ is triangular, Algorithm QN requires $4k'n + 2k'^2 + O(k') + O(n)$ multiplications. If u is available from a previous computation, then only $2k'n + 2k'^2 + O(k') + O(n)$ multiplications are necessary.

If the quasi-Newton step is not within the trust region, then one needs to compute $g_k^T B_k g_k$, which can be determined using the intermediate quantities of Algorithm QN as follows.

ALGORITHM CBG.

1. Solve $\begin{bmatrix} E_k^{1/2} & 0 \\ -Z^{(k)}E_k^{-1/2} & J_k \end{bmatrix} t = \begin{bmatrix} u_1 \\ \delta u_2 \end{bmatrix}$.
2. Partition t as $\begin{bmatrix} t_1 \\ t_2 \end{bmatrix} \begin{matrix} k' \\ k' \end{matrix}$.
3. $\nabla f(x_k)^T B_k \nabla f(x_k) = \delta \nabla f(x_k)^T \nabla f(x_k) + t_1^T t_1 - t_2^T t_2$.

Because E_k is diagonal and $Z^{(k)}$ is triangular, the cost of Algorithm GBG is only $k'^2 + O(n) + O(k')$ multiplications. There is no nk' term in the operation count because we were able to use partial results from Algorithm QN. This same saving does not occur with the standard Cholesky factorization of B_k and is one of the major benefits of this approach.

Once a step has been determined one needs to adjust the trust radius τ and update the representations of B_k and H_k. The calculation of the trust region depends on (2.3). For (2.3), we get

$$s_{k+1}^T B_k s_{k+1} = -\alpha^2 \eta^T g_k + \beta^2 g_k^T B_k g_k - 2\alpha\beta g_k^T g_k, \tag{2.13}$$

since $B_k \eta = -g_k$. Computing the right-hand side of (2.13) requires at most $O(n)$ multiplications because either $\beta = 0$ or $g_k^T B_k g_k$ has been precomputed.

2.1. Updating B_k and H_k. At the kth iteration the computation of H_{k+1} requires the matrix products $Y_k^T y_{k+1}$ and $S_k^T y_{k+1}$, i.e., $A_k^T y_{k+1}$. Since $y_{k+1} = \nabla f(x_{k+1}) - \nabla f(x_k)$, $A_k^T y_{k+1} = A_k^T \nabla f(x_{k+1}) - A_k^T \nabla f(x_k) = A_k^T \nabla f(x_{k+1}) - u_k$, where u_k was used and computed in Algorithm QN. If one computes $u_{k+1} = A_{k+1}^T \nabla f(x_{k+1})$ and saves it for the next application of Algorithm QN, then the work involved in Algorithm QN is reduced by $2nk'$ multiplications.

In general, to have a representation of C_{k+1}^{-1} in the formula for B_{k+1}, one needs $Z^{(k+1)}$ and $S_{k+1}^T S_{k+1}$, which in turn requires $Y_k^T s_{k+1}$ and $S_k^T s_{k+1}$, i.e., $A_k^T s_{k+1}$. Normally one would expect that the computation of $A_k^T s_{k+1}$ would require $2nk'$ multiplications. However, from (2.4), if $\alpha = 0$, we get $A_k^T s_{k+1} = \beta u_k$, which requires only $O(n)$ multiplications, and if $\alpha \neq 0$, from (2.11) and Algorithm QN, we see that

$$A_k^T s_{k+1} = (\beta - \alpha\delta^{-1}) u_k - \alpha A_k^T A_k \begin{bmatrix} -\delta^{-1} w \\ p \end{bmatrix}, \tag{2.14}$$

which costs $4k'^2$ multiplications because the matrix $A_k^T A_k$ is readily available.

We now turn our attention to the computation of J_k in (2.10). Byrd, Nocedal, and Schnabel [4] assume that k' is so small compared to n that the $O(k'^3)$ work involved with computing J_k from scratch each iteration will be dwarfed by the $O(nk')$ operations involved in multiplying $A_k^T \nabla f(x_k)$. In our work we will not make that assumption and we will try to avoid operations that involve $O(k'^3)$ multiplications.

If $k < \hat{k}$, at the next iteration, S_k will gain a column and $Z^{(k)}$, a strictly lower triangular matrix, will gain a row. Thus the matrix V_k in (2.10) will gain another row and column, and we find that V_{k+1} will have the form

$$V_{k+1} = \begin{bmatrix} V_k & v \\ v^T & \beta \end{bmatrix}, \tag{2.15}$$

where $v^T = \delta s_{k+1}^T S_k + z^T E_k^{-1} Z^{(k)^T}$, $\beta = \delta s_{k+1}^T s_{k+1} + z^T z / e_{k+1}$, and z^T is the kth row of $Z^{(k)}$. Similarly, J_{k+1} has the form

$$J_{k+1} = \begin{bmatrix} J_k & 0 \\ p^T & \gamma \end{bmatrix}, \tag{2.16}$$

where $p = J_k^{-1} v$ and $\gamma^2 = \beta - (p^T p)$. Updating J requires $k^2 + O(k)$ multiplications.

When $k > \hat{k}$, information must first be removed from S_k before the above algorithm is begun. Downdating essentially means that S_k would lose its first column and the lower triangular matrix $Z^{(k)}$ its first column, z_1. Removing the z_1 information from J entails forming the Cholesky factorization $\tilde{J}\tilde{J}^T$ of the matrix

$$\tilde{V} = V_k - e_1^{-1} z_1 z_1^T \tag{2.17}$$

for which five algorithms are given in [10]. In our implementation we have used the second one which requires $1.5k'^2 + O(k')$ multiplications.

An algorithm for removing the S information from J_k can be derived by noticing that J_k is the transpose of the R matrix in the QR decomposition of the matrix

$$F = \begin{bmatrix} \delta^{1/2} S_k \\ E_k^{-1/2} Z^{(k)^T} \end{bmatrix}, \tag{2.18}$$

since $V_k = F^T F$. Note that F initially has the structure

$$\begin{bmatrix} x & x & x & x \\ .. & . & . & . \\ x & x & x & x \\ - & - & - & - \\ & x & x & x \\ & & x & x \\ & & & x \end{bmatrix}. \tag{2.19}$$

Since the QR decomposition of F is given by

$$F = Q_F \begin{bmatrix} J_k^T \\ 0 \end{bmatrix}, \tag{2.20}$$

eliminating the first column of S_k is equivalent to removing the first column of F in (2.19). This, in turn, is equivalent to deleting the first row from J_k, which is also the first row of $\tilde{J}$. This leaves us with a matrix $\tilde{J}$ of the form

$$\begin{bmatrix} x & x & & & \\ x & x & x & & \\ .. & . & . & x & \\ .. & . & . & . & x \\ x & x & x & x & x \end{bmatrix}. \tag{2.21}$$

Because $\tilde{J}^T$ is part of the QR decomposition of the F matrix, we are free to apply orthogonal transformations to its rows (i.e., to the columns of (2.21)) to return it to triangular form. Givens transformations applied successively to the columns of (2.21) in the planes $(1,2),(2,3),\ldots,(k',k'-1)$ can be used to return it to lower triangular form. Assuming four multiplications per Givens transformation, this part of the algorithm requires $2k'^2 + o(k')$ multiplications.

Table 2.1 below summarizes the operation count for the whole algorithm according to the type of step taken. In the rest of the paper we will call this approach ATA, indicating that it is based on $A^T A$.

Computing a decomposition of J from scratch each time rather than performing downdates and updates incurs a cost of $k'^2/2 + k'^3/6$ multiplications per iteration, which is efficient only if $k' < 18$ and there are many iterations.

TABLE 2.1
Multiplication counts of each iteration depending on the type of step.

Task	Quasi-Newton step	Not quasi-Newton step
Compute QN step Compute $g^T Bg$	$2nk' + 2k'^2$	$2k'n + 2k'^2$ k'^2
Updating $A^T A$	$2nk' + 4k'^2$	$2nk' + 4k'^2$
Downdating J Updating J	$3.5k'^2$ k'^2	$3.5k'^2$ k'^2
Total	$4nk' + 10.5k'^2$	$4nk' + 11.5k'^2$

The line search quasi-Newton algorithms in [4] require at least $4k'n + O(k'^2)$ multiplications, so that the line search schemes and ATA have the same leading term. Those in [4] based on updating B compute a decomposition of V_k in (2.9) rather than updating the decomposition as detailed in (2.15)–(2.21), and thus incur an additional cost of $k'^3/6$ multiplications per iteration, which is consequential when, say, $k > 18$. Thus the algebraic overhead costs for the trust region approach per iteration are not greater than those for the line search approach. On any given problem any comparison between the two limited memory algorithms rests mainly on the effectiveness of the line search scheme versus the trust region mechanism. In our experience the line search approach usually requires more function evaluations per iteration but may require fewer iterations than the trust region method if the quadratic model used by the trust region approach is not "trustworthy." In the limited memory approach, where some information is discarded, the reliability of the model tends to be more sensitive to the value of δ than the nonlimited memory trust region algorithm and may require higher values of k than the limited memory line search approach.

2.2. Using the QR decomposition. Throughout our description of the algorithm we have encountered submatrices of $A_k^T A_k$. Rather than forming this matrix explicitly, one could take a hint from Nazareth [13] and form the QR decomposition of the $n \times 2k'$ matrix A_k of rank m, given by

$$A_k = Q_k \begin{bmatrix} R_k \\ 0 \end{bmatrix}, \tag{2.22}$$

where Q_k is an orthogonal matrix and R_k is an $m \times 2k'$ upper trapezoidal matrix. Rather than storing Q_k, one would store only its first m columns, which we will call $\hat{Q}_k$. The main advantage of using (2.22) is numerical stability. The A_k matrix itself would not have to be stored. Whenever it is needed for matrix-vector multiplication, (2.22) would be used. If one denoted the first k' columns of R_k as R_Y, then $Y^T Y$ in (2.11) could be written simply as $R_Y^T R_Y$. The product would never be formed, but to produce $z = Y^T Y x$ one would set $v = R_Y x$ followed by $z = R_Y^T v$. Actually, it would be more practical to interleave the columns of A_k to form the matrix $\tilde{A}_k$. At each iteration $\tilde{A}_k$ would gain two columns corresponding to s_k and y_k. Using the Gram–Schmidt procedure with reorthogonalization, updating the QR decomposition of $\tilde{A}_k$ would cost $2nm(l+1) + O(n) + O(k')$ multiplications, where l represents the number of reorthogonalization steps [5]. If $m = 2k'$, this cost is about double the cost of updating $A_k^T A_k$.

The cost of updating and downdating the decomposition depends on the rank of the matrix $\tilde{A}_k$. We will show that for $k \le 2\hat{k}$, the rank m_k of A_k satisfies $m_k \le k+1$.

Our proof depends on the matrix Φ_k, which has the structure

$$\Phi_k = [s_1, g_1, s_2, g_2, \ldots, s_k, g_k]. \tag{2.23}$$

THEOREM 1. *Assuming the search direction is a linear combination of the steepest descent direction and the quasi-Newton direction, then for $k < 2\hat{k}$ the rank m'_k of Φ_k satisfies $m'_k \leq k+1$.*

Proof. The proof is by induction on k. For $k = 1$, the matrix Φ_k has at most two linearly independent columns.

Assume $k < 2\hat{k}$ and Φ_k has rank m'_k, where $m'_k \leq k+1$. The vectors s_{k+1} and g_{k+1} are appended to Φ_k to form Φ_{k+1}. Now from (2.4) and the intermediate quantities of Algorithm QN, we see that

$$s_{k+1} = (\beta - \alpha\delta^{-1})g_k + \delta^{-1}\alpha Y_k w - \alpha S_k p. \tag{2.24}$$

Since $y_k = g_k - g_{k-1}$, the columns of Y_k are linear combinations of the columns of Φ_k. Thus s_{k+1} is a linear combination of the columns of Φ_k, the rank of $(\Phi_k | s_{k+1})$ is m'_k, and the rank of Φ_{k+1} is at most $m'_k + 1$. From our induction hypothesis we get that $m'_{k+1} \leq k+2$, thus proving the theorem. ∎

From Theorem 1, we can prove the following theorem.

THEOREM 2. *Assuming the search direction is a linear combination of the steepest descent direction and the quasi-Newton direction, then for $k < 2\hat{k}$ the rank m_k of $\tilde{A}_k$ satisfies $m_k \leq k+1$.*

Proof. Consider the matrix Ψ_k a $2k \times 2k$ matrix that has the form

$$\Psi_k = \begin{bmatrix} 1 & \psi & & & & & & & \\ & 1 & -1 & & & & & & \\ & & 1 & & & & & & \\ & & & 1 & & -1 & & & \\ & & & & 1 & & & & \\ & & & & & 1 & & -1 & \\ & & & & & & 1 & & \\ & & & & & & & 1 & \end{bmatrix},$$

where $\psi = -||g_0||/\tau$. Note that $s_1 = \psi^{-1} g_0$. The matrix $\tilde{A}_k = \Phi_k \Psi_k$ and hence has the same rank as Φ_k, since Ψ_k is nonsingular. ∎

In practice one would work with and save $\hat{Q}_k$, the first m_k columns of the Q matrix from the QR of Φ_k. Since $\hat{Q}_k$ has at most $k+1$ columns, $\hat{Q}_k$ occupies about half the space of A_k, which is why Theorems 1 and 2 consider $k < 2\hat{k}$ rather than $k \leq \hat{k}$. Applying A_k to a vector x using (2.22) requires $kn + k^2 + O(k) + O(n)$ multiplications rather than $2nk$ multiplications if A itself were used. Updating (2.22) requires $2nk(l+1) + O(n+k)$ multiplications, and, without reorthogonalization, about the same amount required to compute $A_k^T A_k$.

Downdating the Gram–Schmidt QR decomposition of $\tilde{A}_k$ is much more expensive than the downdating in (2.10) because, as shown in [5], here one has to apply transformations to $\hat{Q}_k$, which means that we will be dealing with $O(nk)$ multiplications rather than $O(k^2)$. Between iterations $\hat{k}$ and $2\hat{k}$, one can cite two reasons for delaying the downdating process. First, if $\hat{k}$ is large enough, one may converge to the solution of the optimization problem before iteration $2\hat{k}$, so that the downdating process might be superfluous. Second, it costs less to wait, assuming that shedding the first two columns each time does not decrease the rank of $\tilde{A}_k$. From Theorem 2 we gather

TABLE 2.2
Multiplication count with several options assuming "combination step."

	Determining step and radius	Updating matrices	Downdating matrices	Total
ATA($k < \hat{k}$)	$2k'n + 3k'^2$	$2k'n + 5k'^2$		$4k'n + 8k'^2$
QR ($k < \hat{k}$)	$k'n + 5k'^2$	$2k'n + 2k'^2$		$3k'n + 7k'^2$
ATA($k \geq \hat{k}$)	$2k'n + 3k'^2$	$2k'n + 5k'^2$	$3.5k'^2$	$4k'n + 11.5k'^2$
QR ($\hat{k} \leq k < 2\hat{k}$)	$kn + 3k'^2 +$ $2k^2$	$2kn + 2k^2$		$3kn + 3k'^2 +$ $4k^2$
QR ($k = 2\hat{k}$)	$4k'n + 7k'^2$	$4k'n + 2k'^2$	$2k'^2n$	$2k'^2n + 8k'n +$ $9k'^2$
QR ($k > 2\hat{k}$)	$4k'n + 7k'^2$	$4k'n + 2k'^2$	$10k'n + 13k'^2$	$18k'n + 22k'^2$

that the matrix $\tilde{R}_k$ (corresponding to $\tilde{A}$) has the structure of (2.25). Downdating each iteration requires a sequence of three-plane Householder transformations that require about five multiplications per transformation. By iteration $2\hat{k}$ one would need $\hat{k}^2$ such transformations, so that the total cost of applying these transformations to $\hat{Q}_k$ is about $5\hat{k}^2n$ multiplications. If one waited until iteration $2\hat{k}$ and eliminated the first $2\hat{k}$ columns of R_k and reduced the next $2\hat{k}$ columns to triangular form, then one would be using longer Householder transformations, and the application of these to $\hat{Q}$ would require at most $2\hat{k}^2n$ multiplications. Thus postponing is cost effective.

$$\begin{bmatrix} x & x & x & x & x & . & x & x & x \\ & x & x & x & x & . & x & x & x \\ & & & x & x & . & x & x & x \\ & & & & & . & x & x & x \\ & & & & & . & x & x & x \\ & & & & & & x & x & x \\ & & & & & & & & x \end{bmatrix}. \tag{2.25}$$

After iteration $2\hat{k}$ downdating costs $10k'n + 10k'^2$ multiplications to apply the transformations to $\hat{Q}_k$ and $\tilde{R}_k$. Referring to Table 2.2 one realizes that downdating swamps all the other algebraic computations. Moreover, the $k'n$ term in the downdating multiplication count refers to vector operations and not to matrix-vector operations, which could be implemented with fast BLAS [3]. For some problems it may make sense just to begin again, throwing out old information, recomputing δ, and taking a steepest descent step. In the next section we will call this option RSQR, for "restart using the QR decomposition."

Table 2.2 provides the cost of the A^TA-based and the QR-based algorithms, assuming both α and β in (2.4) are nonzero. The major lesson one gleans from the table is that downdating using the QR decomposition is expensive.

3. Numerical examples. The reduced storage algorithms defined in section 2 were inserted into Dennis and Mei's MINOP [7] code restricted to the BFGS update. These modified codes were applied to two problems to determine if in practice there were significant differences between the algorithms. The codes were run on a four-processor SGI machine and terminated when the gradient decreased to 1×10^{-6}.

The first problem is the journal bearing problem discussed in [1] without the nonnegativity constraints. In that problem we set the eccentricity ϵ to .1 and b to 1. For the journal bearing problem, function and gradient evaluations account for about

TABLE 3.1
Function evaluations and time (sec) for the journal bearing problem.

	$n = 400$		$n = 1600$		$n = 2500$	
	Fn. eval.	Time	Fn. eval.	Time	Fn. eval.	Time
MINOP, $\hat{k} = n$	35	6.7				
LBFGS, $\hat{k} = 50$	55	.54	96	3.6	111	8.4
RSQR, $\hat{k} = 50$	43	.57	70	3.7	87	7.2
ATA, $\hat{k} = 50$	43	.52	68	4.0	85	8.5
LBFGS, $\hat{k} = 25$	55	.55	96	3.9	111	7.2
RSQR, $\hat{k} = 25$	41	.48	117	5.3	166	11.1
ATA, $\hat{k} = 25$	43	.53	65	3.3	88	6.8
LBFGS, $\hat{k} = 10$	55	.50	102	3.6	111	6.0
RSQR, $\hat{k} = 10$	60	.64	163	6.4	238	15.1
ATA, $\hat{k} = 10$	42	.45	71	3.0	109	6.6

half the total computation time in the limited memory codes we tested.

In Table 3.1 we compare the original MINOP code algorithm to ATA and RSQR and LBFGS, a limited memory line search BFGS code discussed in [11]. For $n = 400$ the decrease in the number of linear algebra computations per iteration with the reduced storage schemes significantly reduces the overall time. This is definitely a situation in which one would want to use these types of schemes over the straight quasi-Newton codes.

From Table 3.1 one can make several observations. There was not a great deal of difference between the timings of the line search routine and ATA. The line search routine tended to take two function evaluations per iteration, while ATA usually took one, so that the number of function evaluations of LBFGS tended to be higher but fewer iterations were required. For this problem the trust region routines were more sensitive to the value of $\hat{k}$ and the smaller the value of $\hat{k}$, the more function evaluations were required. This was particularly true of the algorithm RSQR that restarted every $\hat{k}$ iterations.

The results for ATA and RSQR were very dependent on the choice of δ. We followed the suggestion of Shanno and Phua [16] and Oren and Spedicato [15] that δ be set initially to $y^T y / s^T y$, where $y = \nabla f(x_1) - \nabla f(x_2)$ and $s = x_1 - x_2$. For restarting RSQR, this formula was used. For this choice of δ, most of the steps were in the quasi-Newton direction. For the problem with $n = 2500$ and $\hat{k} = 25$, we multiplied the initial δ by 10 and found that most of the steps were combination steps, and for the same accuracy, 204 function evaluations were needed and the computation required 19 seconds. When we divided the initial δ by 10, again most of the steps were combination steps, the number of function evaluations increased to 333, and the computation required 22 seconds.

The second example was a small ($n = 167$) version of a molecular dynamics problem involving silicon and oxygen, which had prompted this research. The problem involved finding "unique" vectors z_i in 3-space, which minimized

$$f = \sum_i \sum_j a_{ij}/d_{ij} + b_{ij}/d_{ij}^6 + c_{ij} e^{(h_{ij} - d_{ij})}, \tag{3.1}$$

where $d_{ij} = ||z_i - z_j||_2^2$ and the a's, b's , c's, and h's were known constants. Unfortunately, there were some negative values of $b_{i,j}$, and for these values, if $z_i = z_j$, then $f = -\infty$. The application had many local minima, and different minima were found

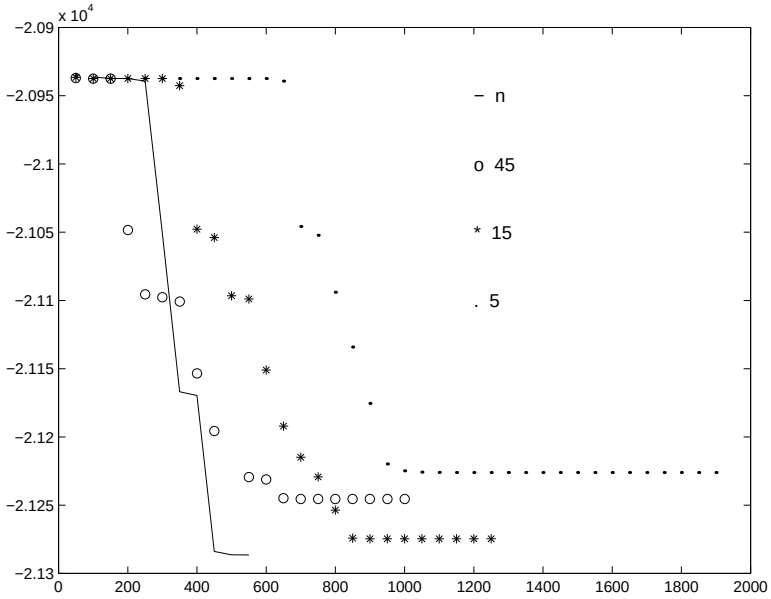

FIG. 3.1. *Number of iterations for various values of $\hat{k}$ for the silicon problem.*

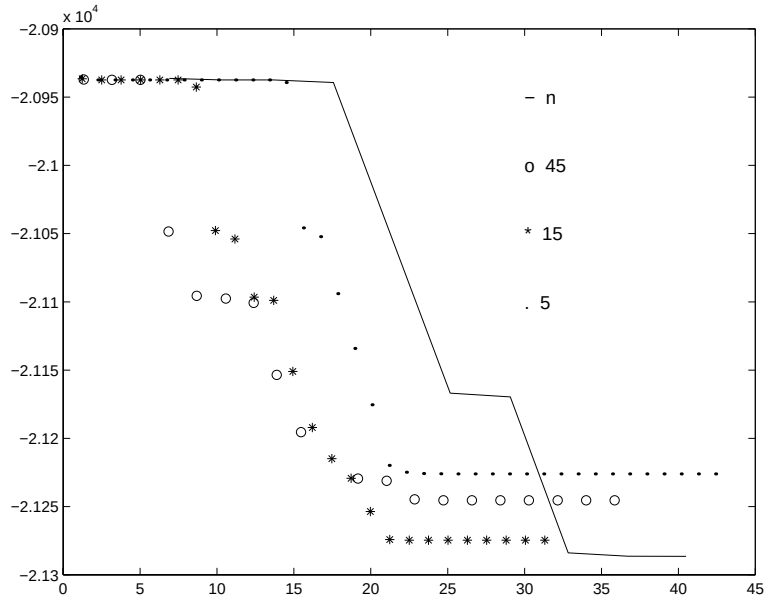

FIG. 3.2. *Times for various values of $\hat{k}$ for the silicon problem.*

as $\hat{k}$ and δ were varied. This made an exact comparison rather difficult.

Figure 3.1 plots the function values versus the number of function evaluations for the original nonreduced storage algorithm to the ATA scheme for several values of $\hat{k}$, and Figure 3.2 compares the time in seconds on our SGI machine.

Analytic gradients were computed along with the function values to take advan-

tage of common subexpressions. The initial value of δ given above was also used here. As expected, the original scheme, which does not try to minimize storage, required the least number of iterations, and the smaller the value of $\hat{k}$, the larger the number of function evaluations. However, in terms of time, all the runs were similar. Increasing δ by a factor of 10 sometimes improved the results and sometimes led to a different minimum. The restarting scheme RSQR tended to require more iterations than the ATA scheme near the local minima. The function evaluation profile and time profile for RSQR with $\hat{k} = 45$ resembled that of ATA with $\hat{k} = 5$. The line search routine found the solution at $-\infty$ and balked whenever the user-specified bound on the line search step was shortened or a penalty term was added to (3.1) to prevent the d_{ij}'s from going to zero.

4. Conclusion. We have shown that one can construct a reduced storage dogleg–trust region quasi-Newton code that not only reduces the storage significantly, but can also reduce the total computation time over the traditional quasi-Newton dogleg–trust region methods on some problems. This scheme is competitive with the limited memory line search program LBFGS. Its algebraic overhead per iteration theoretically is either the same or less than limited memory line search program algorithms depending on how the line search routine is implemented. The same updating techniques that were described here could be introduced in some line search algorithms to lower their operation count. In practice, the reduced storage dogleg approach is very sensitive to the settings of some of its parameters and its performance is dependent on whether the model is "trustworthy."

Acknowledgment. I would like to thank the referees for their insightful comments and careful reading of the manuscript.

REFERENCES

[1] B. Averick, R. Carter, and J. Moré , *The Minpack-2 Test Problem Collection*, ANL-MCS-TM-150, Argonne National Laboratory, Argonne, IL, 1991.

[2] A. Buckley and A. LeNir *QN-like variable storage conjugate gradients*, Math. Programming, 27 (1983), pp. 155–175.

[3] J. Burke and A. Wiegmann, *Notes on limited memory BFGS updating in a trust-region framework*, SIAM J. Optim., submitted.

[4] R. Byrd, J. Nocedal, and R. Schnabel, *Representation of quasi-Newton matrices and their use in limited memory methods*, Math. Programming, 63 (1994), pp. 129–156.

[5] J. Daniel, W. Gragg, L. Kaufman, and G.W. Stewart, *Stable algorithms for updating the Gram–Schmidt QR factorization*, Math. Comp., 30 (1976), pp. 772–795.

[6] J. Dennis, D. Gay, and R. Welsch, *Algorithm* 611 *subroutines for unconstrained minimization using a model/trust-region approach*, ACM Trans. Math. Software, 9 (1983), pp. 503–524.

[7] J. Dennis and H. Mei, *An unconstrained optimization algorithm which uses function and gradient values*, J. Optim. Theory Appl., 28 (1979), pp. 455–480.

[8] J. Dennis and R. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice–Hall, Englewood Cliffs, NJ, 1983; reprinted as Classics Appl. Math. 16, SIAM, Philadelphia, PA, 1996.

[9] J. Dongarra, J. DuCroz, S. Hammerling, and R. Hanson, *An extended set of Fortran basic linear algebra solvers*, ACM Trans. Math. Software, 14 (1989), pp. 1–17.

[10] P. Gill, G. Golub, W. Murray, and M. Saunders, *Factorized variable metric methods for unconstrained optimization*, Math. Comp., 30 (1976), pp. 796–811.

[11] D. Liu and J. Nocedal, *On the limited memory BFGS method for large scale optimization*, Math. Programming, 45 (1989), pp. 503–528

[12] J. Moré and D. Sorensen, *Computing a trust region step*, SIAM J. Sci. Stat. Comput., 4 (1983), pp. 553–572.

[13] L. Nazareth, *The method of successive affine reduction for nonlinear minimization*, Math. Programming, 35 (1986), pp. 97–109.

[14] J. Nocedal, *Updating quasi-Newton matrices with limited storage*, Math. Comp., 35 (1980), pp. 773–782.
[15] S. Oren and E. Spedicato, *Optimal conditioning of self-scaling and variable metric algorithms*, Math Programming, 10 (1976), pp. 70–90.
[16] D. Shanno and K. Phua, *Matrix conditioning and nonlinear optimization*, Math. Programming, 14 (1978), pp. 145–160.