Efficient and Stable Computation of Quasi-Newton Updates*†

Vasile Sima

Computer Process Control Laboratory
Research Institute for Informatics
8-10 Averescu Avenue,
71316 Bucharest 1
ROMANIA

Abstract: Quasi-Newton techniques are frequently used for the numerical solution of unconstrained and linearly or nonlinearly constrained optimization problems. The key computational step of these techniques is the updating of a symmetric positive definite matrix after a symmetric rank-two modification, involving an addition and a subtraction of dyads. Some inefficiencies and numerical difficulties may arise mainly due to the subtraction operation. The paper discusses efficient and stable quasi-Newton updates using various orthogonal and unitary transformations.

Keywords: Cholesky factorization, downdating, matrix modifications, optimization, orthogonal transformations, quasi-Newton methods, updating.

Vasile Sima was born at Lita, Romania, on the 21st of October, 1949. He graduated the Polytechnical Institute of Bucharest in Control Engineering in 1972, and the Department of Mathematics at the University of Bucharest, in 1978. He obtained his doctoral degree in Control Engineering (adaptive control) at the Polytechnical Institute of Bucharest, in 1983. Since 1972 he has held several research positions at the Research Institute for Informatics in Bucharest. He is senior research worker and the Secretray of the Scientific Board of the institute. He is also an associate professor at the Polytechnical Institute of Bucharest. Dr. Vasile Sima published more than 70 scientific papers (more than 30 of them were published in international journals and symposia proceedings). He is also co-author of the books: "Computer-Aided Optimization Practice" and "Adaptive and Flexible Control of Industrial Processes", and author of the book "New Methods in Applied Mathematics", all of them written in Romanian. His research interests include automatic control theory, adaptive and optimal control, computer-aided control systems design, nonlinear programming, numerical linear algebra and scientific computations.

1. Introduction

Quasi-Newton methods are frequently used for solving various nonlinear optimization problems, including unconstrained, and linearly or nonlinearly constrained optimization problems. Their main advantage over the Newton methods is the reduction by an order of size of the computational burden per iteration. In addition, no second-order derivatives of problem functions need be evaluated. The compensation for these advantages is that superlinear rather than quadratic convergence rate can be proven for quasi-Newton methods.

Most algorithms solving nonlinear optimization problems make, at each iteration, a line search in the space of the variables, $x \in \mathbf{R}^n$, in order to minimize the objective function $F(x)$ and achieve feasibility, that is satisfy the equality and/or inequality constraints defined by some vector function $c(x)$. Quasi-Newton methods construct (on each iteration) a positive definite approximation H of the Hessian matrix involved in the algorithm used. For unconstrained problems, H is an approximation to the Hessian of $F(x)$ at the current iterate; for nonlinearly constrained optimization problems, H may be an approximation to the Hessian of the Lagrangian function, $F(x) - \lambda^T c(x)$. These approximations are updated from an iteration to the next one using various updating formulas, like *Powell-Symmetric-Broyden* (PSB), *Davidon-Fletcher-Powell* (DFP), or *Broyden-Fletcher-Goldfarb-Shanno* (BFGS). A

*Presented at the Fourth SIAM Conference on Optimization, May 11-13, 1992, Chicago, IL, USA.

†Research for this article was supported in part by a grant from the International Research & Exchanges Board (IREX), with funds provided by the Andrew W. Mellon Foundation, the National Endowment for the Humanities, and the U.S. Department of State, and by the Research Institute for Informatics (Grant 668c/1992 — Science and Instruction Ministry). None of these organizations is responsible for the views expressed.

general discussion on quasi-Newton methods can be found, for instance, in (Gill, Murray and Wright, 1981 [6]). Specific quasi-Newton formulas are used, for example, in (Moré, Garbow and Hillstrom, 1980 [10]; Powell, 1982, 1987, 1989 [11] - [13]; Sima, 1990 [14]), for the solution of various optimization problems.

The BFGS formula enjoys an increasing popularity among the quasi-Newton updates. This formula, as well as other quasi-Newton formulas, can be written in the simplified form

$$\tilde{H} = H + \alpha u u^T - \beta v v^T, \tag{1}$$

where $H = H^T \in \mathbf{R}^{n \times n}$, $u, v \in \mathbf{R}^n, \alpha, \beta \in (0, \infty)$, and $H > 0, \tilde{H} > 0$. Updating schemes based on directly applying (1) can determine numerical difficulties, such as loss of symmetry and/or loss of positive definiteness in $\tilde{H}$ in finite precision arithmetic. Such difficulties are especially current in the computations performed in recursive estimation, or adaptive control and signal processing applications, where formulas like (1) are used for updating the covariance matrix or the so-called information matrix. (See, for instance, Ferng, Golub and Plemmons (1991) [4], for a discussion related to the usage of sliding window methods in recursive least squares estimation.) Better numerical results can be expected when the Cholesky factor R of H, rather than H itself, is updated.

Five methods for updating the Cholesky factor of a positive definite matrix after a symmetric rank-one modification have been described by Gill and co-workers (1975) [5]. Recently, Bartels and Kaufman (1989) [1] have indicated the way each method of the five brings about a single application of a rank-two method. For some methods, this involves a new Householder transformation technique designed to annihilate elements of two vectors at once using a rank-one correction of the identity matrix. Their numerical experiments on scalar, vector and shared-memory multiple-instructions multiple-data machines render as more economical a rank-two update rather than two rank-one updates. However, they do not consider pipelining two applications of the rank-one algorithms, which is possible under certain circumstances.

Without loss of generality, α and β in (1) can be taken as being unity. (Otherwise, they can be incorporated in u and v, respectively.) Hence, (1) can be rewritten as

$$\tilde{R}^T \tilde{R} = R^T R + u u^T - v v^T, \tag{2}$$

where R and $\tilde{R}$ are (nonsingular) upper triangular matrices — the Cholesky factors of R and $\tilde{R}$, respectively. Algorithms for computing the new Cholesky factor $\tilde{R}$ after H has been modified by an addition and a subtraction of dyads as in (2) will be discussed in the next sections. The addition of a dyad is called *updating*, and the subtraction is called *downdating*. The term "updating" will also be used for the whole modifying process.

Computational schemes for updating the Cholesky factor typically involve sequences of orthogonal plane rotations annihilating the update vector u^T. Let $G(i, j)$ denote a *standard plane rotation* in the plane (i, j), that annihilates the j-th element and modifies the i-th element of a given vector x. The matrix $G(i, j)$ is the identity matrix, except for the elements (i, i), (i, j), (j, i), (j, j), which define the matrix $G = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$, where $c^2 + s^2 = 1$, and c, s can essentially be chosen as

$$r = (a^2 + b^2)^{1/2},\ c = a/r,\ s = b/r,$$

where $a \triangleq x_i$, $b \triangleq x_j$.

The downdating computations may require the usage of *hyperbolic rotations*. They can be defined as quite similar to standard plane rotations. As shown in Section 3, a hyperbolic rotation is used to annihilate the second element of a vector $(a, \mathrm{i}b)^T$, where $a, b \in \mathbf{R}$ and $\mathrm{i} \triangleq (-1)^{1/2}$. Assuming that $| a | \geq | b |$, it is possible to use a matrix like G in the preceding paragraph, where

$$r = (a^2 - b^2)^{1/2},\ c = a/r,\ s = \mathrm{i}b/r.$$

Clearly, a hyperbolic rotation is a unitary matrix. It will be shown later on in Section 3 that only real arithmetic is necessary to accomplish the required computations.

For convenience, the calculations for a plane rotation can be represented by

$$(c, s, r) = rotg(a, b);,$$

which means that the rotation is defined by parameters c and s, and, if applied, it transforms the vector $(a, b)^T$ into the vector $(r, 0)^T$. If a may be overwritten by r, then one writes

$$(c, s, a) := rotg(a, b);.$$

(The symbol ":=" is also used in other situations to denote overwriting.) When r is not important to be mentioned, it may be omitted. Similar conventional notations are used for hyperbolic rotations. The corresponding algorithm is denoted *roth*.

It is worth mentioning that on implementing various algorithms, the formulas above should be improved, to avoid arithmetic exceptions, like overflows or underflows, and to increase the robustness and accuracy of computations. Standard routines from the BLAS collection [9] can be used.

The notation "$i = n : l : m$", used in some algorithms below, means that the integer index i takes the values $n, n + l, \ldots, m$, where $m \bmod l = n$, and l can be positive or negative. The updated quantities will be marked by tilde. $\|\cdot\|$ denotes the Euclidean vector norm. Other notation conventions will be defined when needed.

2. Updating Using Plane Rotations

Algorithms which efficiently update the Cholesky factor of a symmetric $n \times n$ matrix H, after H has been modified by adding and/or subtracting a symmetric rank-one matrix are presented. Sequences of plane rotations can be used.

Consider first the updating of factorization after adding a rank-one matrix. Hence, given the Cholesky factor R of H, the problem is that of obtaining the updated Cholesky factor $\tilde{R}$ of the matrix $\tilde{H} = H + aa^T$, where a is a real n-vector. Therefore,

$$\tilde{R}^T\tilde{R} = R^TR + aa^T = [R^T \;\; a]\begin{bmatrix} R \\ a^T \end{bmatrix}. \tag{3}$$

Let G be an orthogonal matrix computed so that

$$G\begin{bmatrix} R \\ a^T \end{bmatrix} = \begin{bmatrix} \tilde{R} \\ 0 \end{bmatrix}. \tag{4}$$

Then, $\tilde{R}$ is the required updated Cholesky factor. Matrix G can be chosen as the product $G_n \cdots G_1$, where $G_i = G(i, n + 1)$ is a standard plane rotation in the plane $(i, n + 1)$. The algorithm can be applied in a column-wise manner [3], as sketched below:

Algorithm *update*(R, a). Given the $n \times n$ upper triangular matrix R and the n-vector a, this algorithm computes the upper triangular matrix $\tilde{R}$ that satisfies $\tilde{R}^T\tilde{R} = R^TR + aa^T$. $\tilde{R}$ is overwritten on R.

1) For $j = 1 : n$,

   1) For $i = 1 : j - 1$, /* Apply the previous rotations on column j */

      1) $t = r_{ij}$; $r_{ij} = c_it + s_ia_j$; $a_j := c_ia_j - s_it$;

   2) $(c_j, s_j, r_{jj}) := rotg(r_{jj}, a_j)$; /* Compute the next rotation */.

Consider now the updating of factorization after subtracting a rank-one matrix. Hence, given the Cholesky factor R of H, the problem is that of obtaining the updated Cholesky factor $\tilde{R}$ of the matrix $\tilde{H} = H - aa^T$, or $\tilde{R}^T\tilde{R} = R^TR - aa^T$. Therefore,

$$R^TR = \tilde{R}^T\tilde{R} + aa^T = [\tilde{R}^T \quad a]\begin{bmatrix} \tilde{R} \\ a^T \end{bmatrix}. \tag{5}$$

Let G be an orthogonal matrix computed so that

$$G^T\begin{bmatrix} \tilde{R} \\ a^T \end{bmatrix} = \begin{bmatrix} R \\ 0 \end{bmatrix}. \tag{6}$$

Then, $\tilde{R}$ is the required updated Cholesky factor. The algorithm for computing $\tilde{R}$ is not so simple as above. Its main steps are listed in the procedure below.

1. Solve the system $R^Ty = a$.
2. If $\|y\| \geq 1$, "Downdating is not possible" and Stop. Else, $\eta \triangleq (1 - \|y\|^2)^{1/2}$.
3. Determine the rotations $G_1, \ldots, G_n$, with G_i in the plane $(n+1, i)$, so that
$$G_1G_2\cdots G_n\begin{bmatrix} y \\ \eta \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$
4. Construct
$$\begin{bmatrix} \tilde{R} \\ \tilde{a}^T \end{bmatrix} = G_1G_2\cdots G_n\begin{bmatrix} R \\ 0 \end{bmatrix}.$$

To justify this procedure, one must prove that $\|y\| < 1$ is a necessary condition for making downdating possible, and that the step 4 gives $\tilde{a} = a$ and an upper triangular matrix $\tilde{R}$. The first assertion follows noting that $R^TR - aa^T = R^T(I_n - yy^T)R$, but $I_n - yy^T$ has a nonpositive eigenvalue $1 - \|y\|^2$, if $\|y\| \geq 1$, and so, by Sylvester's theorem of inertia, the matrix $R^TR - aa^T$ is not positive definite. (Indeed, if we write $(I_n - yy^T)x_i = \lambda_i x_i$, $i = 1:n$, where λ_i and x_i are the i-th eigenvalue and the corresponding eigenvector, we obtain $x_i - (y^Tx_i)y = \lambda_i x_i$. Certainly, we can take $\lambda_i = 1$, $x_i \perp y$, $i = 1 : n-1$, where $x_i^Tx_j = 0$, $i \neq j$, and the n-th eigenvalue follows from the condition $\sum_{i=1}^n \lambda_i = \mathrm{tr}(I_n - yy^T) = n - \|y\|^2$.) At step 3, each rotation G_i annihilates the i-th element and modifies the $(n+1)$-th element of the vector $[y^T, \eta]^T$. Note that the final value of the last element is 1, due to the orthogonality of G_i, $i = 1 : n$, and the definition of η. Moreover, the fact that $\tilde{R}$ is upper triangular derives from the structure of rotations G_i, and from the order under which they are applied. Let r_j and $\tilde{r}_j$ denote the j-th columns of R and $\tilde{R}$, respectively. Since the last $n-j$ elements of r_j are zero, and unchanged by $G_{j+1}, \ldots, G_n$, it follows that

$$\begin{bmatrix} \tilde{r}_j \\ \tilde{a}_j \end{bmatrix} = G_1\cdots G_n\begin{bmatrix} r_j \\ 0 \end{bmatrix} = G_1\cdots G_j\begin{bmatrix} r_j \\ 0 \end{bmatrix}. \tag{7}$$

Finally, since $G = G_1\cdots G_n$ is orthogonal, the relations at steps 1, 3 and 4 give

$$a = [R^T \quad 0]\begin{bmatrix} y \\ \eta \end{bmatrix} = [\tilde{R}^T \quad \tilde{a}]\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \tilde{a}.$$

The downdating algorithm can be applied in a column-wise manner [3], as sketched below:

Algorithm *downdate*(R, a). Given the $n \times n$ upper triangular matrix R and the n-vector a, this algorithm computes the upper triangular matrix $\tilde{R}$, such that $\tilde{R}^T\tilde{R} = R^TR - aa^T$. $\tilde{R}$ is overwritten on R.

1) For $j = 1 : n$, /* Solve $R^T y = a$ in s */

   1) $s_j = (a_j - \sum_{i=1}^{j-1} r_{ij} s_i)/r_{jj}$;

2) $\eta = \|s\|$; If $\eta \geq 1$, "Downdating is not possible" and Stop; $\eta := (1 - \eta^2)^{1/2}$;

3) $(c_i, s_i, \eta) := rotg(\eta, s_i)$, $i = n : -1 : 1$; /* Determine rotations */

4) For $j = 1 : n$, /* Apply rotations on R */

   1) $\alpha = 0$;

   2) For $i = j : -1 : 1$,

      1) $t = c_i \alpha + s_i r_{ij}$; $r_{ij} := c_i r_{ij} - s_i \alpha$; $\alpha = t$;.

Note that vector y in step 1 of the procedure discussed above is computed by Algorithm *downdate* in the vector s.

Now, the computation of $\tilde{R}$ in (2) can be simply accomplished using the following algorithm.

Algorithm *qNrotg*(R, u, v). Given the $n \times n$ upper triangular matrix R and the n-vectors u, v, this algorithm computes the upper triangular matrix $\tilde{R}$ that satisfies (2). $\tilde{R}$ is overwritten on R.

1) *update*(R, u);

2) *downdate*(R, v);

Asymptotically, algorithm *qNrotg* requires $4.5n^2$ multiplications, $2.5n^2$ additions and $2n$ square roots. The calculations in *update* and step 1 in *downdate* can be performed simultaneously.

3. Updating Using Plane and Hyperbolic Rotations

Consider now an algorithm that efficiently updates the Cholesky factor of a symmetric matrix H, after H has been modified by subtracting a symmetric dyad. A sequence of hyperbolic rotations can be used. The algorithm is similar to Algorithm *update*. Indeed, an upper triangular matrix $\tilde{R}$ should be computed such that

$$\tilde{R}^T \tilde{R} = R^T R - aa^T = [R^T \quad \mathrm{i}a] \begin{bmatrix} R \\ \mathrm{i}a^T \end{bmatrix}. \tag{8}$$

Let G be a unitary matrix computed so that

$$G \begin{bmatrix} R \\ \mathrm{i}a^T \end{bmatrix} = \begin{bmatrix} \tilde{R} \\ 0 \end{bmatrix}. \tag{9}$$

Then, $\tilde{R}$ is the updated Cholesky factor. Matrix G can be chosen as the product $G_n \cdots G_1$, where G_i is a hyperbolic rotation in the plane $(i, n+1)$. Clearly, G_i is defined by

$$r = (r_{ii}^2 - a_i^2)^{1/2}, \; c_i = r_{ii}/r, \; s_i = \mathrm{i}a_i/r.$$

Therefore, by premultiplying the matrix $\begin{bmatrix} R \\ \mathrm{i}a^T \end{bmatrix}$ by G_i, it follows that

$$\tilde{r}_{ii} = r; \; \tilde{r}_{ij} = (r_{ii} r_{ij} - a_i a_j)/r, \; \mathrm{i}\tilde{a}_j = \mathrm{i}(a_j r_{ii} - a_i r_{ij})/r, \; j = i+1 : n. \tag{10}$$

These results allow a simplification of the notation used. Indeed, let us redefine $s_i = a_i/r$, while omitting the purely imaginary factor. Then, (10) becomes

$$\tilde{r}_{ii} = r; \; \tilde{r}_{ij} = c_i r_{ij} - s_i a_j, \; \tilde{a}_j = c_i a_j - s_i r_{ij}, \; j = i+1 : n. \tag{11}$$

Consequently, only real arithmetic is used throughout, the hyperbolic rotation G_i can be written exactly as a plane rotation, and the updating formulas are almost identical to those corresponding to an addition of a dyad. The only differences made are in the minus sign in the relation for computing r, used to define the rotation and obtain the modified element $\tilde{r}_{ii}$, and in that for calculating $\tilde{r}_{ij}$.

Therefore, the downdating algorithm based on hyperbolic rotations parallels Algorithm *update*. (We have only to change a sign in the step 1.1.1 and to replace *rotg* by *roth* in the step 1.2.) For this reason, it is not given explicitly here. Below, *update* and hyperbolic downdating algorithms are combined. To improve the efficiency, they are interlaced, since each rotation i acts only on the i-th and $(n+1)$-th rows, and the hyperbolic rotation can be used immediately after applying the corresponding plane rotation.

Algorithm *qNroth*(R, u, v). Given the $n \times n$ upper triangular matrix R and the n-vectors u, v, this algorithm computes the upper triangular matrix $\tilde{R}$ that satisfies (2). $\tilde{R}$ is overwritten on R.

1) For $j = 1 : n$,

  1) For $i = 1 : j-1$, /* Apply the previous rotations on column j */

    1) $t = r_{ij};\ s = c_i t + s_i u_j;$

    2) $r_{ij} = c_i^h s - s_i^h v_j;\ u_j := c_i u_j - s_i t;\ v_j := c_i^h v_j - s_i^h s;$

  2) $(c_j, s_j, r_{jj}) := rotg(r_{jj}, u_j);$ /* Compute the next plane rotation */

  3) $(c_j^h, s_j^h, r_{jj}) := roth(r_{jj}, v_j);$ /* Compute the next hyperbolic rotation */.

Asymptotically, this algorithm requires $4n^2$ multiplications, $2n^2$ additions and $2n$ square roots. The use of multiprocessing can be very profitable.

4. Updating Using Other Transformations

Plane and hyperbolic rotations have been used in the preceding sections for updating the Cholesky factor involved in the calculations. Note that the computation of a rotation G and its application to two n-vectors involve about $4n$ multiplications, $2n$ additions and one square root. Instead of standard rotations, modified rotations can be used, with less computational effort. Assume that matrix $X \in \mathbf{R}^{2\times n}$, which the rotation G should be applied to, is available in the factored form, $X = D^{1/2}Y$, where $D = \mathrm{diag}(d_1^{1/2}, d_2^{1/2})$ is a diagonal matrix. Refactorizing $GD^{1/2}$ as $\tilde{D}^{1/2}M$, it follows that

$$GX = GD^{1/2}Y = \tilde{D}^{1/2}MY = \mathrm{diag}(\tilde{d}_1^{1/2}, \tilde{d}_2^{1/2})MY. \tag{12}$$

The right hand side of (12) provides an updated factorization of the product GX. The key point is that matrix M can be chosen so that two of its elements are 1. This removes $2n$ multiplications when forming the product MY. The computations can be arranged in a stable manner, and so that only the squares of the "weighting factors" $d_i^{1/2}$ should appear in matrix M. (See [9], for details.) This allows that square roots in the calculations are avoided and negative weights, for downdating, are dealt with. Since repeated applications of the modified rotations can lead the weighting factors to overflow or underflow, scaling is used to adjust these factors when needed. In such situations, M can have three or even four elements differing from 1, so that computational savings might result.

The usage of modified rotations raises no conceptual difficulty and, therefore, these rotations are not considered in the sequel.

Householder transformations can also be used instead of rotations. For a non-zero vector $t \in \mathbf{R}^n$, the corresponding *Householder transformation* is the symmetric elementary matrix $T = I_n - tt^T/\beta$, where $\beta = \frac{1}{2} \| t \|^2$. A Householder matrix is clearly orthogonal and hence it preserves the Euclidean length of vectors. For any two nonzero vectors x and y of equal length, there is a Householder transformation T that transforms x in y, since $Tx = x - (t^T x/\beta)t = y$, if t is appropriately chosen to be parallel to $x - y$; moreover, it can be noticed that the transformed vector is the difference between

the original vector and a multiple of the Householder vector t, hence x and y will have identical components in the positions where t has null components. In our application, y must have certain zero elements. For instance, choosing $t_1 = r + x_1$; $t_i = x_i$, $i = 2 : n$, where $r = \text{sign}(x_1) \parallel x \parallel$, we have $Tx = -re_1$, e_1 being the first coordinate vector. (In this case, we have $\beta = 1/[r(r + x_1)]$.)

Of special interest in our calculations are modified Householder transformations of order two. A *modified Householder transformation* of order p is defined by $T = I_p + zw^T$ $(= I_p - tt^T/\beta)$, where the p-vectors z and w are chosen as $z = -t/r$, $w = t/(r + x_1)$ (r being defined in the preceding paragraph), so that given the p-vector x, $x \neq 0$, it follows as above that $Tx = -re_1$. The calculation and application of modified Householder transformations can reduce the computational burden for $p \leq 3$. For large p, storing two vectors, z and w, is not convenient and so, the standard Householder transformations are preferred.

Note that the application of a modified Householder transformation of order two on two vectors of length n involves $3n$ multiplications and additions, so that using it is theoretically better than using rotations. Our limited numerical experience has revealed that this is not true in practice for certain machines like FORMOX 286 (IBM-PC compatible) microcomputers. Vectors with elements randomly generated by a uniform distribution in the range (0,1) have been used.

5. Updating after General Rank-Two Modifications

We will generalize the problem dealt with in the preceding sections, intending to update efficiently the Cholesky factorization of H after a symmetric rank-two modification described by

$$\tilde{H} = H + Z^T B Z,\ B = B^T \in \mathbf{R}^{2\times 2},\ Z \in \mathbf{R}^{2\times n},\ H > 0,\ \tilde{H} > 0. \tag{13}$$

We shall describe a technique to solve the problem. Let $V \in \mathbf{R}^{2\times n}$ be the solution of the matrix equation $VR = Z$. Let Q be an $n \times n$ orthogonal matrix such that $Q^T V^T \triangleq \hat{V}^T = \begin{bmatrix} \bar{V} \\ 0 \end{bmatrix}$, where $\bar{V} \in \mathbf{R}^{2\times 2}$. Note that Q^T can be chosen as the product of plane rotations $(\prod_{i=1}^{n-2}(G'(i+1, i+2)G(i, i+1)))G(n-1, n)$, where $G(j-1, j)$ and $G'(j-1, j)$ annihilate the j-th element of the first and second column of V^T, respectively. Then,

$$\tilde{H} = R^T(I_n + V^T B V)R = R^T Q(I_n + \hat{V}^T B \hat{V})Q^T R. \tag{14}$$

Now, since $\tilde{H} > 0$, the Sylvester's theorem of inertia implies $I_n + \hat{V}^T B \hat{V} > 0$, and also, $I_2 + \bar{V} B \bar{V}^T > 0$. Therefore, $I_2 + \bar{V} B \bar{V}^T$ has a Cholesky factorization, $I_2 + \bar{V} B \bar{V}^T = \bar{R}^T \bar{R}$, where $\bar{R} \in \mathbf{R}^{2\times 2}$ is upper triangular. Denoting

$$J = \begin{bmatrix} \bar{R} & 0 \\ 0 & I_{n-2} \end{bmatrix},\ M = Q^T R,\ \tilde{M} = JM, \tag{15}$$

it follows that

$$\tilde{H} = R^T Q J^T J Q^T R = M^T J^T J M = \tilde{M}^T \tilde{M}. \tag{16}$$

Taking into account the specially chosen structure of Q^T, it is easy to see that M and $\tilde{M}$ have zeros below the second subdiagonal. Let $\tilde{Q}$ be an orthogonal matrix so that $\tilde{Q}^T \tilde{M} = \tilde{R}$ is upper triangular. Matrix $\tilde{Q}^T$ can be chosen as $\prod_{i=1}^{n-1} M_i$, where M_i is a modified Householder transformation that zeroes $\tilde{m}_{i+1,i}$ and $\tilde{m}_{i+2,i}$ for $i = 1 : n-2$, and $\tilde{m}_{i+1,i}$ for $i = n-1$. Therefore, $\tilde{H} = \tilde{R}^T \tilde{R}$ is the required Cholesky factorization and has been obtained in $0(n^2)$ operations.

Asymptotically, the algorithm described above requires a total of $11n^2 + 0(n)$ operations, which indicates that one would not expect a large speedup on a scalar machine. On a parallel machine, one can solve the upper triangular system with two right-hand sides simultaneously. Reorganizing the computations gives much room for parallelization and vectorization.

When B is indefinite and rank-two, one can combine the ideas for a positive update with those for a negative update. Since B is symmetric, one can find an eigendecomposition of B as $B = QDQ^T$,

where Q is an orthogonal matrix and D is diagonal. Since B is indefinite, one may assume that $d_{11} > 0$ and $d_{22} < 0$. Denoting

$$Y = Z^T Q \begin{bmatrix} d_{11}^{1/2} & \\ & (-d_{22})^{1/2} \end{bmatrix} \triangleq [u, v], \tag{17}$$

and $\hat{H} = H + uu^T$, then it follows that $\tilde{H} = \hat{H} - vv^T$. Clearly, one could use Algorithms *qNrotg* or *qNroth* for obtaining the Cholesky factor of $\tilde{H}$. The algorithm can be summarized as follows:

1) Form the eigendecomposition of B, $B = QDQ^T$.

2) Compute Y in (17).

3) Use $qNroth(R, u, v)$.

Numerous other cheaper techniques for updating the Cholesky factor after general rank-two modifications are described in [1]. Some of these techniques deal with positive or negative definite B matrices only.

Goldfarb (1976) [7] has also considered some methods for updating the Cholesky factorization of the matrices produced using variable metric methods for minimizing functions. In particular, he deals with updating the factor of the matrix

$$\tilde{H} = (I_n + vu^T)H(I_n + uv^T), \tag{18}$$

which is slightly less general than formula (13), since the rank-two correction term has one nonnegative and one nonpositive eigenvalue, while the signs of the corresponding eigenvalues in (13) are not restricted. (Indeed, it is easy to see that for (18), we have $B = \begin{bmatrix} 0 & 1 \\ 1 & u^T H u \end{bmatrix}$, so that the product of eigenvalues of B is -1.) Goldfarb's method is based on rewriting (18) as

$$\tilde{H} = R^T(I_n + zw^T)(I_n + wz^T)R,$$

where $R^T z = v$, $R^T w = Hu$ (or $w = Ru$), and R is an upper triangular matrix so that $R^T R = H$. An orthogonal matrix Q is then found such that $Q(I_n + wz^T) = \hat{R}$, where $\hat{R}$ is an upper triangular matrix whose elements above the diagonal are essentially given by $\hat{r}_{ij} = f_i^T g_j$, f_i and g_j being vectors of length two. Obtaining $\hat{R}$ requires $0(n)$ operations, and multiplying $\hat{R}$ by R for obtaining the Cholesky factor of $\tilde{H}$ requires $2n^2 + 0(n)$ operations. We have to add the cost for computing z and w.

6. Numerical Experiments

The updating algorithm of the Cholesky factor of the Hessian approximation by a BFGS formula, based on standard plane rotations, has been extensively used for solving nonlinearly constrained optimization problems. Powell's constrained variable metric method [11], incorporating the watchdog technique in [2], was employed. Over 25 test problems in [17] and [8] have been solved using various values for certain key parameters, such as the requested accuracy and the "gain factor" introduced for avoiding incompatible quadratic programming subproblems. Details about the solution techniques and the numerical results are presented in [15], [16].

Hundreds of quasi-Newton updates were involved in these experiments. No numerical difficulties were encountered for reasonable values of the requested accuracy, `ACC`. (For certain test problems, `ACC` was chosen near the relative machine accuracy.) All the computed results were highly accurate. Table 1 gives a synthesis of the results for various test problems in [8] and [17]. EX641 – EX647 are the Examples 6.4.1 – 6.4.7, respectively, from [17]. EX647M and EX647L are modifications of Ex. 6.4.7, by considering only one equality constraint, or only linear constraints, respectively. The

other problems are from [8]. Problem TP84 was scaled. The columns **ITER** and **NFEV** in the table give the number of iterations and the number of function and gradient evaluations, respectively. The columns entitled "x accuracy" and "F accuracy" indicate the number of correct decimal figures in the computed x and F, respectively, compared to the results in [8], for which the solutions are known. The execution time is expressed in seconds and has been determined using a double precision function subprogram, that uses the operating system routines which provide the hour, minute, second and hundredths of second. An increase in speed was obtained by using the current information about the active constraints when passing from an iteration to the next one. A nonstandard value of the gain factor (10^{36}), for avoiding incompatible subproblems was employed.

Table 1: Numerical results for nonlinearly constrained optimization problems.

Problem	ACC	Time (sec.)	ITER	NFEV	x accuracy	F accuracy
EX641	10^{-12}	1.70	12	14		
EX642	10^{-10}	3.07	14	16		
EX643	10^{-14}	5.88	8	9		
EX646	10^{-16}	1.21	9	9		
EX647L	10^{-12}	10.21	6	6		
EX647	10^{-12}	6.87	4	4		
EX647M	10^{-14}	25.59	15	19		
TP51	10^{-16}	0.50	3	5	16	30
TP56	10^{-13}	4.18	13	14	10	14
TP57	10^{-13}	1.60	13	15	7	10
TP66	10^{-15}	0.83	9	9	10	10
TP84*	10^{-7}	2.25	9	9	12	10
TP100	10^{-12}	5.10	20	27	9	9
TP113	10^{-13}	9.95	14	18	8	10
TP117	10^{-11}	41.53	19	19	8	10

7. Conclusions

Various algorithms for efficient and stable computation of quasi-Newton updates have been described. They are based either on two rank-one updates, or on one rank-two update. Such algorithms can be applied for solving unconstrained and linearly or nonlinearly constrained optimization problems. They are also useful in the related areas of recursive parameter estimation, adaptive filtering, prediction and control, or signal processing.

REFERENCES

1. BARTELS, R., KAUFMAN, L., **Cholesky factor updating techniques for rank 2 matrix modifications**. SIAM J. MATRIX ANAL. APPL. 10 (1989), pp. 557–592.
2. CHAMBERLAIN, R.M., LEMARÉCHAL, C., PEDERSEN, H.C., POWELL, M.J.D., **The watchdog technique for forcing convergence in algorithms for constrained optimization**. MATH. PROG. STUDY 16 (1982), pp. 1–17.
3. DONGARRA, J.J., BUNCH, J.R., MOLER, C.B., STEWART, G.W., **LINPACK Users Guide**. SIAM PUBLICATIONS, Philadelphia (1979).
4. FERNG, W.R., GOLUB, G.H., PLEMMONS, R.J., **Adaptive Lanczos methods for recursive condition estimation**. NUMER. ALGORITHMS 1 (1991), pp. 1–20.
5. GILL, P.E., GOLUB, G.H., MURRAY, W., SAUNDERS, M.A., **Methods for modifying matrix factorizations**. MATH. COMP. 28 (1975), pp. 505–535.

6. GILL, P.E., MURRAY, W., WRIGHT, M.H., **Practical Optimization**. ACADEMIC PRESS, New York (1981).
7. GOLDFARB, D., **Factorized variable metric methods for unconstrained optimziation**. MATH. COMP. 30 (1976), pp. 796–811.
8. HOCK, W., SCHITTKOWSKI, K., **Test Examples for Nonlinear Programming Codes**. Lecture Notes in Economics and Mathematical Systems (187). SPRINGER-VERLAG (1981).
9. LAWSON, C., HANSON, R., KINCAID, D., KROGH, F., **Basic linear algebra subprograms for Fortran usage**. ACM TRANS. MATH. SOFTW. 5 (1979), pp. 308–321.
10. MORÉ, J.J., GARBOW, B.S., HILLSTROM, K.E., **User's Guide for MINPACK-1**. Report ANL-80-74, Applied Math. Division, ARGONNE NATIONAL LABORATORY, Argonne, Illinois (1980).
11. POWELL, M.J.D., **VMCWD: A Fortran subroutine for constrained optimization**. Report DAMTP 1982/NA4, UNIVERSITY OF CAMBRIDGE (1982).
12. POWELL, M.J.D., **Updating conjugate directions by the BFGS formula**. MATH. PROG. 38 (1987), pp. 29–46.
13. POWELL, M.J.D., **TOLMIN: A Fortran package for linearly constrained optimization calculations**. Report DAMTP 1989/NA2, UNIVERSITY OF CAMBRIDGE (1989).
14. SIMA, V., **A software package for constrained optimization with applications to control problems**. STUDIES AND RESEARCHES IN COMPUTERS AND INFORMATICS 3 (1990), pp. 117–127.
15. SIMA, V., **Algorithm PODQP: Solving positive definite quadratic programming problems**. STUDIES IN INFORMATICS AND CONTROL 1 (1992a), pp. 63–71.
16. SIMA, V., **Algorithm VMCWDM: Solving nonlinearly constrained optimization problems**. STUDIES IN INFORMATICS AND CONTROL 1 (1992b), pp. 151–165.
17. SIMA, V., VARGA, A., **Computer Aided Optimization Practice**. EDITURA TEHNICĂ, Bucureşti (in Romanian) (1986).