KYBERNETIKA — VOLUME 49 (2013), NUMBER 2, PAGES 224–235

RECURSIVE FORM OF GENERAL LIMITED MEMORY VARIABLE METRIC METHODS

Ladislav Lukšan and Jan Vlček

In this report we propose a new recursive matrix formulation of limited memory variable metric methods. This approach can be used for an arbitrary update from the Broyden class (and some other updates) and also for the approximation of both the Hessian matrix and its inverse. The new recursive formulation requires approximately $4mn$ multiplications and additions per iteration, so it is comparable with other efficient limited memory variable metric methods. Numerical experiments concerning Algorithm 1, proposed in this report, confirm its practical efficiency.

Keywords: unconstrained optimization, large scale optimization, limited memory methods, variable metric updates, recursive matrix formulation, algorithms

Classification: 49K35, 90C06, 90C47, 90C51

1. INTRODUCTION

Limited memory variable metric methods, introduced in [11], are intended for solving large scale unconstrained optimization problems, where the objective function $F : R^n \to R$ is continuously differentiable, bounded from below and has unknown or dense Hessian matrix. They are usually realized in a line search framework, so their iteration step has the form

$$x_{i+1} = x_i + t_i s_i \tag{1}$$

for $i \in \mathcal{N}$ ($\mathcal{N}$ is the set of positive integers), where $s_i = -H_i g_i$ is the direction vector ($g_i = g(x_i)$ is the gradient of the objective function at the point x_i and H_i is a positive definite approximation of the inverse Hessian matrix in the ith iteration) and $t_i > 0$ is the step-length, which is taken to satisfy the weak Wolfe conditions

$$F_{i+1} - F_i \leq \varepsilon_1 t_i s_i^T g_i, \tag{2}$$

$$s_i^T g_{i+1} \geq \varepsilon_2 s_i^T g_i, \tag{3}$$

with $0 < \varepsilon_1 < 1/2$ and $\varepsilon_1 < \varepsilon_2 < 1$ (where $F_i = F(x_i)$, $g_i = g(x_i)$ and $F_{i+1} = F(x_{i+1})$, $g_{i+1} = g(x_{i+1})$). We concentrate our attention to the limited memory variable metric methods from the Broyden class [9], but in the Section 2 we show that our recursive algorithm can be also applied to methods from the Davidon class [3].

Let $0 < \bar{m} < n$, $i \in \mathcal{N}$ and $m = \min(\bar{m}, i)$. Limited memory variable metric methods from the Broyden class use direction vectors $s_1 = -g_1$ and $s_{i+1} = -H_{i+1}g_{i+1}$, $i \in \mathcal{N}$, where matrix $H_{i+1} \stackrel{\Delta}{=} H^i_{i+1}$ is obtained from a sparse positive definite (usually scaled unit) matrix H^i_{i-m+1} by means of m updates

$$H^i_{j+1} = H^i_j + U^i_j M^i_j (U^i_j)^T, \tag{4}$$

$i - m + 1 \le j \le i$, where matrices $U^i_j = [d_j, H^i_j y_j]$ and M^i_j are chosen to satisfy quasi-Newton conditions $H^i_{j+1} y_j = d_j$, where $y_j = g_{j+1} - g_j$, $d_j = x_{j+1} - x_j$, $i - m + 1 \le j \le i$ (we use upper index i, to signify the relation to the ith iteration). Formula (4) can be written in the form

$$H^i_{j+1} = H^i_j + \frac{1}{b_j} d_j d_j^T - \frac{1}{a^i_j} H^i_j y_j (H^i_j y_j)^T + \frac{\eta^i_j}{a^i_j} \left(\frac{a^i_j}{b_j} d_j - H^i_j y_j \right) \left(\frac{a^i_j}{b_j} d_j - H^i_j y_j \right)^T, \tag{5}$$

where $a^i_j = y_j^T H^i_j y_j$, $b_j = y_j^T d_j$ and η^i_j is a free parameter. Setting $\eta^i_j = 0$, $\eta^i_j = 1$ and $\eta^i_j = b_j/(b_j - a^i_j)$, we obtain the DFP, the BFGS and the Rank-1 updates, respectively. Note that the BFGS update is the most efficient one from these basic updates.

An advantage of limited memory variable metric methods described in this paper is the fact that they can be realized in the way which requires (for n large) approximately $4mn$ multiplications and additions for the direction determination. Phrase approximately $4mn$ means that this number significantly dominates over additional required operations. For example, if $n = 1000$ and $m = 5$, then $4mn = 20000$, whereas $m^3 = 125$. There are two commonly used basic approaches: the recursive vector formulation based on the Strang recurrences [10] and the explicit matrix formulation proposed in [2]. To simplify the notation in the subsequent considerations, we will assume without the loss of generality that $i \le \bar{m}$. Then matrices (4) and (5) do not depend on the upper index, which can be omitted.

The first approach is applicable only in case all matrices H_j, $1 \le j \le i$, are obtained by the BFGS update (in fact there exists other possible updates realizable in this way, see [12], but they do not belong to the Broyden class). The recursive vector formulation of the limited memory BFGS method is based on the pseudo-product form: if $\eta_j = 1$, formula (5) can be written in the form

$$H_{j+1} = V_j^T H_j V_j + \frac{1}{b_j} d_j d_j^T, \qquad V_j = I - \frac{1}{b_j} y_j d_j^T. \tag{6}$$

Using this formula recursively, we obtain

$$H_{i+1} = \left(\prod_{j=1}^{i} V_j \right)^T H_1 \left(\prod_{j=1}^{i} V_j \right) + \sum_{k=1}^{i} \frac{1}{b_k} \left(\prod_{j=k+1}^{i} V_j \right)^T d_k d_k^T \left(\prod_{j=k+1}^{i} V_j \right).$$

Note that matrix H_{i+1} need not be stored, since vector $s_{i+1} = -H_{i+1}g_{i+1}$ can be obtained by two (Strang) recurrences. First we set $u_{i+1} = -g_{i+1}$ and compute numbers σ_j and vectors u_j, $i \ge j \ge 1$, by the backward recurrence

$$\sigma_j = \frac{d_j^T u_{j+1}}{b_j}, \qquad u_j = u_{j+1} - \sigma_j y_j. \tag{7}$$

Then we set $v_1 = H_1 u_1$ and compute vectors v_{j+1}, $1 \le j \le i$, by the forward recurrence

$$v_{j+1} = v_j + \left(\sigma_j - \frac{y_j^T v_j}{b_j} \right) d_j. \tag{8}$$

Finally we set $s_{i+1} = v_{i+1}$.

The use of the Strang recurrences (7)–(8) is the oldest (and simplest) possibility for implementing the limited memory BFGS method. As it was already mentioned, this approach is applicable only if all matrices H_j, $1 \le j \le i$, are obtained by the BFGS update. This disadvantage reveals when we need to update matrix $B_{i+1} = H_{i+1}^{-1}$. It follows from the duality (see [9]) that the Strang recurrences can be used only in case all matrices B_j, $1 \le j \le i$, are obtained by the DFP update. But the limited memory DFP method is much worse than the limited memory BFGS method, so this way is unsuitable.

The second approach is based on the fact that matrix H_{i+1}, obtained by recursive application of i updates of the form (4) to matrix H_1, can be written in the form

$$H_{i+1} = H_1 + \tilde{U}_i \tilde{M}_i \tilde{U}_i^T, \tag{9}$$

where $\tilde{U}_i = [d_1 - H_1 y_1, \ldots, d_i - H_1 y_i]$ and $\tilde{M}_i$ is a square matrix of order m for the Rank-1 update or $\tilde{U}_i = [d_1, \ldots, d_i, H_1 y_1, \ldots H_1 y_i]$ and $\tilde{M}_i$ is a square matrix of order $2m$ otherwise. For the basic updates (DFP, BFGS and Rank-1), the matrix $\tilde{M}_i$ can be expressed in the explicit form. Especially matrix H_{i+1}, obtained by recursive application of i BFGS updates to matrix H_1, can be written in the form

$$H_{i+1} = H_1 + [D_i, H_1 Y_i] \begin{bmatrix} (R_i^{-1})^T (C_i + Y_i^T H_1 Y_i) R_i^{-1}, & -(R_i^{-1})^T \\ -R_i^{-1}, & 0 \end{bmatrix} [D_i, H_1 Y_i]^T, \tag{10}$$

where $D_i = [d_1, \ldots, d_i]$, $Y_i = [y_1, \ldots, y_i]$, R_i is the i-dimensional upper triangular matrix such that $(R_i)_{kl} = d_k^T y_l$, $k \le l$, $(R_i)_{kl} = 0$, $k > l$, and C_i is the i-dimensional diagonal matrix such that $(C_i)_{kk} = d_k^T y_k$ (see [2]). There exists a similar formula for matrix H_{i+1}, obtained by recursive application of i DFP updates to matrix H_1 (see [2]). Using the duality relation between the DFP and the BFGS updates, we can determine the matrix B_{i+1} obtained by recursive application of i BFGS updates to matrix B_1. The resulting matrix can be written in the form

$$B_{i+1} = B_1 - [Y_i, B_1 D_i] \begin{bmatrix} -C_i, & (L_i - C_i)^T \\ L_i - C_i, & D_i^T B_1 D_i \end{bmatrix}^{-1} [Y_i, B_1 D_i]^T, \tag{11}$$

where L_i is the i-dimensional lower triangular matrix such that $(L_i)_{kl} = d_k^T y_l$, $k \ge l$, $(L_i)_{kl} = 0$, $k < l$. The fact that we can use the inverse BFGS updates is very advantageous, since it allows us to implement variable metric trust region methods and methods for constrained optimization, which apply variable metric updates to the part of the KKT matrix.

In this paper, we investigate a modification of the second approach. In Section 2, we propose a new recursive matrix formulation of limited memory variable metric methods.

This approach can be used for both matrices H_{i+1} and B_{i+1} and for an arbitrary update from the Broyden class. Our recursive formulation requires approximately $4mn$ multiplications and additions for the direction determination, so it is comparable with the other approaches mentioned in this paper. At the end of Section 2, we demonstrate that the recursive matrix formulation can be used for some other variable metric updates. As an example, we have chosen the Davidon class of variable metric updates proposed in [3] and reformulated in [6]. Section 3 contains results of numerical experiments which indicates that our approach is competitive with known limited memory variable metric methods.

2. THE RECURSIVE MATRIX FORMULATION

Let us assume that matrix H_{i+1} is obtained from matrix $H_1 = \lambda_i I$ by i updates of the form

$$H_{j+1} = H_j + U_j M_j U_j^T, \qquad 1 \le j \le i \tag{12}$$

(see (4)), where $U_j = [d_j, H_j y_j]$ and

$$M_j = \begin{bmatrix} \alpha_j, & \beta_j \\ \beta_j, & \gamma_j \end{bmatrix}.$$

We seek the expression

$$H_{i+1} = H_1 + \bar{U}_i \bar{M}_i \bar{U}_i^T, \tag{13}$$

where $\bar{U}_i = [d_1, H_1 y_1, \ldots, d_i, H_1 y_i]$ and $\bar{M}_i$ is a square matrix of order $2m$. This formula is very similar to (9). For rank two updates, matrices $\bar{U}_i$ and $\tilde{U}_i$ differ only by orders of its columns. Note that the choice $H_1 = \lambda_i I$ (where usually $\lambda_i = d_i^T y_i / y_i^T y_i$) is essential for our considerations leading to the algorithm described below. This choice is used in the rest of this paper.

Theorem 2.1. Let matrix H_{i+1} be obtained from matrix H_1 by i updates of the form (12). Then (13) holds with matrix $\bar{M}_i$ obtained recursively in such a way that $\bar{M}_1 = M_1$ and

$$\bar{M}_j = \begin{bmatrix} \bar{M}_{j-1} + \gamma_j \, z_{j-1} z_{j-1}^T, & \beta_j \, z_{j-1}, & \gamma_j \, z_{j-1} \\ \beta_j \, z_{j-1}^T, & \alpha_j, & \beta_j \\ \gamma_j \, z_{j-1}^T, & \beta_j, & \gamma_j \end{bmatrix}, \qquad 2 \le j \le i, \tag{14}$$

where

$$z_{j-1} = \bar{M}_{j-1} \bar{r}_{j-1}, \qquad \bar{r}_{j-1} = \bar{U}_{j-1}^T y_j. \tag{15}$$

P r o o f . We prove this theorem by induction. Assume that

$$H_j = H_1 + \bar{U}_{j-1} \bar{M}_{j-1} \bar{U}_{j-1}^T \tag{16}$$

for some index $2 \le j < i$. Relation (16) holds for $j = 2$ by (12) since $\bar{U}_1 = U_1$ and $\bar{M}_1 = M_1$. Substituting (16) into (12) and using the fact that

$$U_j = [d_j, H_j y_j] = \left[d_j, \, H_1 y_j + \bar{U}_{j-1} \bar{M}_{j-1} \bar{U}_{j-1}^T y_j\right] = \left[d_j, \, H_1 y_j + \bar{U}_{j-1} z_{j-1}\right]$$

by (15) and (16), we can write

$$\begin{aligned}
H_{j+1} &= H_1 + \bar{U}_{j-1}\bar{M}_{j-1}\bar{U}_{j-1}^T + \left[d_j,\, H_1 y_j + \bar{U}_{j-1} z_{j-1}\right] M_j \left[d_j,\, H_1 y_j + \bar{U}_{j-1} z_{j-1}\right]^T \\
&= H_1 + \bar{U}_{j-1}\,\bar{M}_{j-1}\,\bar{U}_{j-1}^T + \alpha_j\, d_j d_j^T \\
&\quad + \beta_j \left(d_j (H_1 y_j)^T + H_1 y_j d_j^T\right) + \beta_j \left(d_j\,(\bar{U}_{j-1} z_{j-1})^T + \bar{U}_{j-1} z_{j-1}\, d_j^T\right) \\
&\quad + \gamma_j\, H_1 y_j (H_1 y_j)^T + \gamma_j \left(H_1 y_j\,(\bar{U}_{j-1} z_{j-1})^T + \bar{U}_{j-1} z_{j-1}\,(H_1 y_j)^T\right) \\
&\quad + \gamma_j\, \bar{U}_{j-1} z_{j-1} z_{j-1}^T \bar{U}_{j-1}^T \\
&= H_1 + \left[\bar{U}_{j-1},\, d_j,\, H_1 y_j\right] \left[\begin{array}{lll} \bar{M}_{j-1} + \gamma_j\, z_{j-1} z_{j-1}^T, & \beta_j\, z_{j-1}, & \gamma_j\, z_{j-1} \\ \beta_j\, z_{j-1}^T, & \alpha_j, & \beta_j \\ \gamma_j\, z_{j-1}^T, & \beta_j, & \gamma_j \end{array}\right] \\
&\qquad \left[\bar{U}_{j-1},\, d_j,\, H_1 y_j\right]^T = H_1 + \bar{U}_j \bar{M}_j \bar{U}_j^T,
\end{aligned}$$

so the induction step is proved. □

Comparing (12) with (5), we can see that

$$\alpha_j = \frac{1}{b_j}\left(\eta_j \frac{a_j}{b_j} + 1\right), \qquad \beta_j = -\frac{\eta_j}{b_j}, \qquad \gamma_j = \frac{\eta_j - 1}{a_j}, \tag{17}$$

where $a_j = y_j^T H_j y_j$ and $b_j = y_j^T d_j$. Using (15) and (16), we obtain

$$a_j = y_j^T H_j y_j = y_j^T (H_1 y_j + \bar{U}_{j-1}\bar{M}_{j-1}\bar{U}_{j-1}^T y_j) = y_j^T H_1 y_j + \bar{r}_{j-1}^T z_{j-1},$$

so value a_j (required for the computation of α_j and γ_j by (17)) can be obtained by using known vectors $\bar{r}_{j-1}$ and z_{j-1}.

So far we have assumed that $1 \le i \le \bar{m}$. Now we describe the construction of matrix $H_{i+1} = \lambda_i I + \bar{U}_i \bar{M}_i \bar{U}_i^T$ in the general case. Let $m = \min(\bar{m}, i)$ and $S_i = \mathrm{diag}(1, \lambda_i, \ldots, 1, \lambda_i)$ (where $\lambda_i > 0$) be a $2m$-dimensional diagonal scaling matrix. Denote

$$\begin{aligned}
\check{U}_{i-1} &= \left[d_{i-m+1}, y_{i-m+1}, \ldots, d_{i-1}, y_{i-1}\right], \\
\check{R}_{i-1} &= \left[\begin{array}{lll} d_{i-m+1}^T y_{i-m+1}, & \ldots & d_{i-m+1}^T y_{i-1} \\ y_{i-m+1}^T y_{i-m+1}, & \ldots & y_{i-m+1}^T y_{i-1} \\ \ldots\ldots\ldots & \ldots & \ldots\ldots\ldots \\ 0, & \ldots & d_{i-1}^T y_{i-1} \\ 0, & \ldots & y_{i-1}^T y_{i-1} \end{array}\right]
\end{aligned}$$

(these matrices are empty for $i = 1$) and

$$\hat{U}_i = [\check{U}_{i-1}, d_i, y_i], \qquad \hat{R}_i = \left[\begin{array}{ll} \check{R}_{i-1}, & \check{U}_{i-1}^T y_i \\ 0, & d_i^T y_i \\ 0, & y_i^T y_i \end{array}\right]. \tag{18}$$

Matrices $\check{R}_{i-1}$ and $\hat{R}_i$ are upper block triangular, where every block contains two rows and one column. Then $\bar{U}_i = S_i\hat{U}_i$ and matrix $\bar{M}_i \stackrel{\Delta}{=} \hat{M}_i^i$ is obtained recursively in such a way that we set

$$\hat{M}_{i-m+1}^i = \begin{bmatrix} \alpha_{i-m+1}^i, & \beta_{i-m+1}^i \\ \beta_{i-m+1}^i, & \gamma_{i-m+1}^i \end{bmatrix} \tag{19}$$

and for $i-m+1 \le j \le i-1$ compute vector $z_j^i = \hat{M}_j^i S_j^i \check{r}_j^i$, where S_j^i is the $2(j-i+m)$ dimensional leading submatrix of S_i and $\check{r}_j^i$ is the $2(j-i+m)$ dimensional vector containing first $2(j-i+m)$ elements of the $(j-i+m)$th column of matrix $\check{R}_{i-1}$, and set

$$\hat{M}_{j+1}^i = \begin{bmatrix} \hat{M}_j^i + \gamma_{j+1}^i\, z_j^i (z_j^i)^T, & \beta_{j+1}^i\, z_j^i, & \gamma_{j+1}^i\, z_j^i \\ \beta_{j+1}^i\, (z_j^i)^T, & \alpha_{j+1}^i, & \beta_{j+1}^i \\ \gamma_{j+1}^i\, (z_j^i)^T, & \beta_{j+1}^i, & \gamma_{j+1}^i \end{bmatrix}. \tag{20}$$

Using matrices obtained by the described way, direction vector s_{i+1} can be determined by the formula

$$s_{i+1} = -H_{i+1}g_{i+1} = -\lambda_i g_{i+1} - \bar{U}_i\bar{M}_i\bar{U}_i^T g_{i+1} = -\lambda_i g_{i+1} - \hat{U}_i S_i \hat{M}_i^i S_i \hat{U}_i^T g_{i+1}. \tag{21}$$

In this case, approximately $6mn$ multiplications and additions are consumed for the direction determination ($2mn$ for the determination of the last column of matrix $\hat{R}_i$ and $4mn$ for the computation of vector s_{i+1} by (21)) and approximately $2mn$ values are stored when n is large. Matrices $\check{U}_i$ and $\check{R}_i$ used in the next iteration are easily obtained from $\hat{U}_i$ and $\hat{R}_i$. If $i < \bar{m}$, then $\check{U}_i = \hat{U}_i$ and $\check{R}_i = \hat{R}_i$. If $i \ge \bar{m}$, then $\check{U}_i$ and $\check{R}_i$ arise from $\hat{U}_i$ and $\hat{R}_i$ after the deletion of the columns and rows depending on vectors with index $i-m+1$. Thus

$$[d_{i-m+1}, y_{i-m+1}, \check{U}_i] = \hat{U}_i, \tag{22}$$

$$\begin{bmatrix} d_{i-m+1}^T y_{i-m+1}, & [d_{i-m+1}^T y_{i-m+2}, \ldots, d_{i-m+1}^T y_i] \\ y_{i-m+1}^T y_{i-m+1}, & [y_{i-m+1}^T y_{i-m+2}, \ldots, y_{i-m+1}^T y_i] \\ 0, & \check{R}_i \end{bmatrix} = \hat{R}_i. \tag{23}$$

The above basic process can be modified in such a way that approximately $2mn$ multiplications and additions are dropped. As one can see from (20), the last column $\hat{r}_i$ of matrix $\hat{R}_i$ is not required for the computation of matrix $\hat{M}_i^i$. Thus we can compute vector $\hat{v}_i = \hat{U}_i^T g_{i+1}$ instead of $\hat{r}_i = \hat{U}_i^T y_i$. Vector $\hat{v}_i$ is then used for the determination of the direction vector by the formula

$$s_{i+1} = -\lambda_i g_{i+1} - \hat{U}_i S_i \hat{M}_i^i S_i \hat{v}_i. \tag{24}$$

After the determination of s_{i+1}, one can compute the first $2(m-1)$ elements of $\hat{r}_i$ using the formula

$$\check{U}_{i-1}^T y_i = \check{U}_{i-1}^T g_{i+1} - \check{U}_{i-1}^T g_i, \tag{25}$$

where vector $\check{U}_{i-1}^T g_{i+1}$ contains the first $2(m-1)$ elements of $\hat{v}_i$ (see (18)) and vector $\check{U}_{i-1}^T g_i$ contains the last $2(m-1)$ elements of $\hat{v}_{i-1}$ (vector $\hat{v}_{i-1}$ is known from the previous iteration). The last two elements $d_i^T y_i$ and $y_i^T y_i$ of $\hat{r}_i$ are computed separately, since they serves for the determination of scaling parameter λ_i.

The above considerations are summarized in the following algorithm.

Algorithm 2.2. Data $\bar{m} < n$, $\underline{\varepsilon} > 0$, $0 < \varepsilon_1 < 1/2$, $\varepsilon_1 < \varepsilon_2 < 1$.

Step 1 Let $\check{U}_0$ and $\check{R}_0$ be empty matrices. Choose starting point $x_1 \in R^n$ and compute quantities $F_1 := F(x_1)$, $g_1 := g(x_1)$. Set $s_1 := -g_1$ and $i := 1$.

Step 2 If $\|g_i\| \leq \underline{\varepsilon}$, terminate the computation, otherwise set $m := \min(\bar{m}, i)$.

Step 3 Determine step-size $t_i > 0$ satisfying conditions (2) – (3) and set $x_{i+1} := x_i + t_i s_i$. Compute new quantities $F_{i+1} := F(x_{i+1})$, $g_{i+1} := g(x_{i+1})$ and set $d_i := x_{i+1} - x_i$, $y_i := g_{i+1} - g_i$. Compute values $d_i^T y_i$, $y_i^T y_i$ and set $\lambda_i := d_i^T y_i / y_i^T y_i$ to define $2m$ dimensional scaling matrix $S_i := \mathrm{diag}(1, \lambda_i, \ldots, 1, \lambda_i)$.

Step 4 Determine matrix $\hat{M}^i_{i-m+1}$ by formula (19). Set $\hat{U}_i := [\check{U}_{i-1}, d_i, y_i]$, $\hat{v}_i := \hat{U}_i^T g_{i+1}$ and $j := i - m + 1$.

Step 5 If $j = i$ go to Step 7.

Step 6 Choose the value of parameter η^i_j appearing in (17). Set $z^i_j := \hat{M}^i_j S^i_j \check{r}^i_j$, where S^i_j is the $2(j - i + m)$ dimensional leading submatrix of S_i and $\check{r}^i_j$ is the $2(j - i + m)$ dimensional vector containing the first $2(j - i + m)$ elements of the $(j - i + m)$th column of matrix $\check{R}_{i-1}$, compute matrix $\hat{M}^i_{j+1}$ by (20), set $j := j + 1$ and go to Step 5.

Step 7 Set $\bar{M}_i := \hat{M}^i_i$ and compute direction vector s_{i+1} by formula (24). Compute vector $\check{U}^T_{i-1} y_i$ by (25) and matrix $\hat{R}_i$ by (18).

Step 8 If $i < \bar{m}$, set $\check{U}_i := \hat{U}_i$ and $\check{R}_i := \hat{R}_i$, otherwise determine $\check{U}_i$ and $\check{R}_i$ by (22) and (23). Set $i := i + 1$ and go to Step 2.

The recursive matrix formulation described above can be used also for some other variable metric updates. We focus our attention on the Davidon class of variable metric methods proposed in [3] and reformulated in [6]. Variable metric methods from this class are generalizations of the Rank-1 method. Applied to the quadratic function, they generate conjugate directions without perfect line search.

Limited memory variable metric methods from the Davidon class generate matrix H_{i+1} from matrix $H_1 = \lambda_i I$ by i updates of the form

$$H_{j+1} = H_j + V_j N_j V_j^T, \qquad 1 \leq j \leq i, \tag{26}$$

where $V_j = [v_j, d_j - H_j y_j]$ and

$$N_j = \begin{bmatrix} \rho_j, & \sigma_j \\ \sigma_j, & \tau_j \end{bmatrix}.$$

Vector v_j is generated recursively to satisfy conditions

$$v_{j+1} \in \mathrm{span}(v_j, d_j - H_j y_j), \qquad v^T_{j+1} y_j = 0 \tag{27}$$

(vector v_{j+1} is a linear combination of vectors v_j, $d_j - H_j y_j$ and is perpendicular to vector y_j). Conditions (27) are satisfied, e. g., if

$$v_{j+1} = y_j^T (d_j - H_j y_j) v_j - y_j^T v_j (d_j - H_j y_j). \tag{28}$$

It can be easily proved, see [6], that the update $H_{j+1} = H_j + V_j N_j V_j^T$, where $V_j = [v_j, d_j - H_j y_j]$, satisfies quasi-Newton condition $H_{j+1} y_j = d_j$, if

$$H_{j+1} = H_j + \frac{(d_j - H_j y_j)(d_j - H_j y_j)^T}{y_j^T (d_j - H_j y_j)} - \frac{\varphi_j v_{j+1} v_{j+1}^T}{y_j^T (d_j - H_j y_j)}, \tag{29}$$

where $\varphi_j = -\det N_j$ is a free parameter and v_{j+1} is the vector determined by formula (28). Thus

$$\rho_j = -\varphi_j y_j^T (d_j - H_j y_j), \qquad \sigma_j = \varphi_j y_j^T v_j, \qquad \tau_j = \frac{1 - \varphi_j (y_j^T v_j)^2}{y_j^T (d_j - H_j y_j)}. \tag{30}$$

Setting $\varphi_j = 0$, we obtain the Rank-1 update which lies in both the Broyden and the Davidon classes. It is important that some updates from the Davidon class generate positive definite matrices, but it is computationally difficult to find a suitable value of parameter φ_j, see [6]. Notice that we have chosen the Davidon class of variable metric updates not for its efficiency, but for the demonstration of the fact that the recursive matrix formulation can be also used for variable metric updates that do not belong to the Broyden class.

Analogously to (13), we seek the expression

$$H_{i+1} = H_1 + \bar{V}_i \bar{N}_i \bar{V}_i^T, \tag{31}$$

where $\bar{V}_i = [v_1, d_1 - H_1 y_1, \ldots, v_i, d_i - H_1 y_i]$ and $\bar{N}_i$ is a square matrix of order $2m$.

Theorem 2.3. Let matrix H_{i+1} be obtained from matrix H_1 by i updates of the form (26). Then (31) holds with matrix $\bar{N}_i$ obtained recursively in such a way that $\bar{N}_1 = N_1$ and

$$\bar{N}_j = \begin{bmatrix} \bar{N}_{j-1} + \tau_j z_{j-1} z_{j-1}^T, & \sigma_j z_{j-1}, & \tau_j z_{j-1} \\ \sigma_j z_{j-1}^T, & \rho_j, & \sigma_j \\ \tau_j z_{j-1}^T, & \sigma_j, & \tau_j \end{bmatrix}, \qquad 2 \le j \le i, \tag{32}$$

where

$$z_{j-1} = \bar{N}_{j-1} \bar{r}_{j-1}, \qquad \bar{r}_{j-1} = \bar{V}_{j-1}^T y_j. \tag{33}$$

P r o o f . We prove this theorem by induction. Assume that

$$H_j = H_1 + \bar{V}_{j-1} \bar{N}_{j-1} \bar{V}_{j-1}^T \tag{34}$$

for some index $2 \le j < i$. Relation (34) holds for $j = 2$ by (26) since $\bar{V}_1 = V_1$ and $\bar{N}_1 = N_1$. Denoting $w_j = d_j - H_1 y_j$, substituting (34) into (26) and using the fact that

$$V_j = [v_j, d_j - H_j y_j] = \left[v_j,\, d_j - H_1 y_j + \bar{V}_{j-1} \bar{N}_{j-1} \bar{V}_{j-1}^T y_j\right] = \left[v_j,\, w_j + \bar{V}_{j-1} z_{j-1}\right]$$

by (33) and (34), we can write

$$\begin{aligned} H_{j+1} &= H_1 + \bar{V}_{j-1}\bar{N}_{j-1}\bar{V}_{j-1}^T + \left[v_j,\, w_j + \bar{V}_{j-1}z_{j-1}\right] N_j \left[v_j,\, w_j + \bar{V}_{j-1}z_{j-1}\right]^T \\ &= H_1 + \bar{V}_{j-1}\,\bar{N}_{j-1}\,\bar{V}_{j-1}^T + \rho_j\, v_j v_j^T \\ &\quad + \sigma_j \left(v_j w_j^T + w_j v_j^T\right) + \sigma_j \left(v_j \left(\bar{V}_{j-1}z_{j-1}\right)^T + \bar{V}_{j-1}z_{j-1}\, v_j^T\right) \\ &\quad + \tau_j\, w_j w_j^T + \tau_j \left(w_j \left(\bar{V}_{j-1}z_{j-1}\right)^T + \bar{V}_{j-1}z_{j-1}\, w_j^T\right) \\ &\quad + \tau_j\, \bar{V}_{j-1}z_{j-1}z_{j-1}^T\bar{V}_{j-1}^T \\ &= H_1 + \left[\bar{V}_{j-1},\, v_j,\, w_j\right] \begin{bmatrix} \bar{N}_{j-1} + \tau_j\, z_{j-1}z_{j-1}^T, & \sigma_j\, z_{j-1}, & \tau_j\, z_{j-1} \\ \sigma_j\, z_{j-1}^T, & \rho_j, & \sigma_j \\ \tau_j\, z_{j-1}^T, & \sigma_j, & \tau_j \end{bmatrix} \\ &\quad \left[\bar{V}_{j-1},\, v_j,\, w_j\right]^T = H_1 + \bar{V}_j\bar{N}_j\bar{V}_j^T, \end{aligned}$$

so the induction step is proved. □

Using (33) and (34), we obtain

$$d_j - H_j y_j = d_j - H_1 y_j + \bar{V}_{j-1}\bar{N}_{j-1}\bar{V}_{j-1}^T y_j = d_j - H_1 y_j + \bar{V}_{j-1}z_{j-1},$$

and

$$y_j^T(d_j - H_j y_j) = y_j^T d_j - y_j^T H_1 y_j + \bar{r}_{j-1}^T z_{j-1}.$$

These quantities are necessary for the determination of vector v_{j+1} by (28) and for the computation of numbers ρ_j, σ_j, τ_j by (32).

3. NUMERICAL EXPERIMENTS AND CONCLUSIONS

Limited memory variable metric methods from the Broyden class were tested by using 68 unconstrained minimization problems with 10000 variables from the collection TEST25 described in [7] and 55 problems with 1000–5000 variables from the collection TEST11 described in [8] (problems 15, 26, 33, 42, 48, 57, 58, 60, 61, 67–70, 79 from TEST25 and 42, 48, 50 from TEST11, which were not solved by any limited memory variable metric method, were excluded). These collections, written in Fortran 77, can be downloaded from `http://www.cs.cas.cz/luksan/test.html` together with reports [7] and [8] (TEST11 contains selected large scale problems from the CUTE collection [1]).

The summary results of our tests are presented in two tables given below, where `NIT` is the total number of iterations, `NFV` is the total number of function evaluations and `TIME` is the total computational time. Note that the total computational time is not always proportional to the total number of function evaluations, since individual test problems have different complexity. Rows of tables correspond to the methods tested: BNS – the BFGS method with explicit matrix formulation (formula (10)), New 1.0 – the BFGS method with recursive matrix formulation (Algorithm 1 with $\eta = 1.0$) and New 0.8 variable metric method with recursive matrix formulation (Algorithm 1 with $\eta = 0.8$). The standard value $\bar{m} = 5$ (the number of VM steps) was chosen in all cases. All these methods were implemented with the same line search subroutine using parameters $\underline{\varepsilon} = 10^{-6}$, $\varepsilon_1 = 0.001$, $\varepsilon_2 = 0.9$ and the unit initial step-size (the step-size

from which the line search is started). In fact, we tested many other methods with various values of the parameter η, but the choice $\eta = 0.8$ gave the best results.

Method	NIT	NFV	TIME
BNS	458225	475693	9:49.65
New 1.0	460369	477298	9:20.44
New 0.8	418447	427429	8:29.01

Test 25

Method	NIT	NFV	TIME
BNS	85535	89975	17.11
New 1.0	87742	92382	17.31
New 0.8	84331	87358	15.96

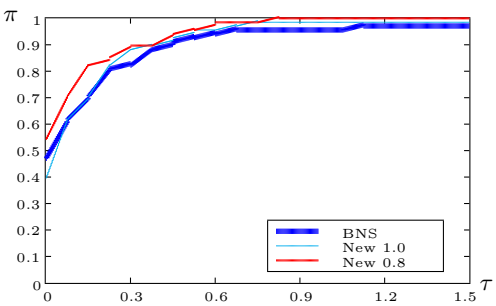
Test 11

For a better demonstration of both the efficiency and the reliability, we compare selected optimization methods by using performance profiles introduced in [4]. The performance profile $\pi_M(\tau)$ is defined by the formula

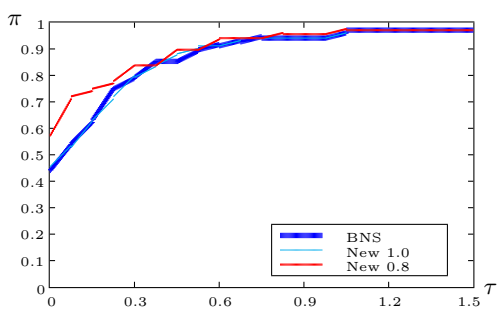
$$\pi_M(\tau) = \frac{\text{number of problems where } \log_2(\tau_{P,M}) \leq \tau}{\text{total number of problems}}$$

with $\tau \geq 0$, where $\tau_{P,M}$ is the performance ratio of the number of function evaluations (or the time) required to solve problem P by method M to the lowest number of function evaluations (or the time) required to solve problem P.

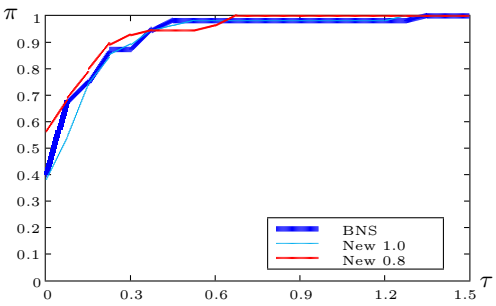
The value of $\pi_M(\tau)$ at $\tau = 0$ gives the percentage of test problems for which the method M is the best and the value for τ large enough is the percentage of test problems that method M can solve. The relative efficiency and reliability of each method can be directly seen from the performance profiles: the higher is the particular curve the better is the corresponding method. The following figures, reveal the performance profiles for tested methods graphically.



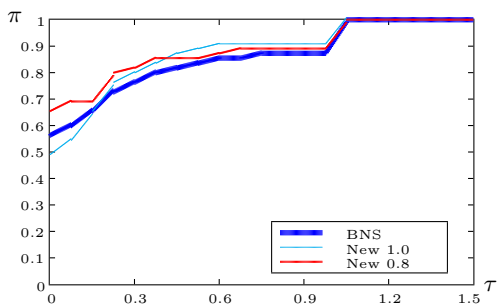
Test 25 – NFV



Test 25 – TIME



Test 11 – NFV



Test 11 – TIME

From the results presented, we can deduce that limited memory variable metric methods with the recursive matrix formulation are at least competitive with standard realizations of limited memory variable metric methods (they use approximately $4mn$ operations for the direction determination as well). Moreover, these results indicate that limited memory variable metric methods from the Broyden class (formula (5)) with values of parameter η different from 1.0, can be more efficient than the limited memory BFGS method. Since we have tested a limited number of simple updates, it is possible that a more suitable choice of parameter η will be found. In this case, such an update will be possible to realize by our recursive formulation approach.

(Received May 10, 2012)

REFERENCES

[1] I. Bongartz, A. R. Conn, N. Gould, and P. L. Toint: CUTE: constrained and unconstrained testing environment. ACM Trans. Math. Software *21* (1995), 123–160.

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

[3] W. C. Davidon: Optimally conditioned optimization algorithms without line searches. Math. Programming *9* (1975), 1–30.

[4] E. D. Dolan and J. J. Moré: Benchmarking optimization software with performance profiles. Math. Programming *91* (2002), 201–213.

[5] S. Hoshino: A formulation of variable metric methods. J. Institute of Mathematics and its Applications *10* (1972), 394–403.

[6] L. Lukšan: Quasi-Newton methods without projections for unconstrained minimization. Kybernetika *18* (1982), 290–306.

[7] L. Lukšan, C. Matonoha, and J. Vlček: Sparse Test Problems for Unconstrained Optimization. Report V-1064, Institute of Computer Science AS CR, Prague 2010.

[8] L. Lukšan, C. Matonoha, and J. Vlček: Modified CUTE Problems for Sparse Unconstrained Optimization. Report V-1081, Institute of Computer Science AS CR, Prague 2010.

[9] L. Lukšan amd E. Spedicato: Variable metric methods for unconstrained optimization and nonlinear least squares. J. Comput. Appl. Math. *124* (2000), 61–95.

[10] H. Matthies and G.Strang: The solution of nonlinear finite element equations. Internat. J. Numer. Methods Engrg. *14* (1979), 1613–1623.

[11] J. Nocedal: Updating quasi-Newton matrices with limited storage. Math. Comput. *35* (1980), 773–782.

[12] J. Vlček and L. Lukšan: Generalizations of the Limited-memory BFGS Method Based on Quasi-product Form of Update. Report V-1060, Institute of Computer Science AS CR, Prague 2009.

Ladislav Lukšan, Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 182 07 Praha 8 and Technical University of Liberec, Hálkova 6, 461 17 Liberec. Czech Republic.

e-mail: luksan@cs.cas.cz

Jan Vlček, Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 182 07 Praha 8. Czech Republic.

e-mail: vlcek@cs.cas.cz