ALTERNATIVE PARAMETER CHOICES FOR MULTI-STEP QUASI-NEWTON METHODS

J.A. Ford and I.A. Moghrabi

Department of Computer Science, University of Essex, Wivenhoe Park,

Colchester, Essex, United Kingdom, CO4 3SQ

Dedicated to Professor Charles Broyden

on the occasion of his sixtieth birthday

Abstract

In a previous paper, Ford and Moghrabi [7] introduced a new, generalized approach to quasi-Newton methods, based on employing interpolatory polynomials which utilize information from the m most recent steps (where standard quasi-Newton methods correspond to m = 1, working only with the most recent step). In these new methods, the iterates were interpolated by a curve in such a way that consecutive points corresponded to a unit-spacing of the parameter defining the curve. In this paper we derive and evaluate some alternative choices for defining the parameter-values which correspond to the iterates on the curve. The experimental results show clearly that such methods can give substantial gains in performance (by comparison with the "unit-spaced" method, which itself yields improvements over standard quasi-Newton methods).

Keywords: Unconstrained optimization, quasi-Newton methods

1. Introduction

Quasi-Newton methods for unconstrained optimisation of an objective function $f(x)$ [where $x \in \mathbb{R}^n$ and f is assumed to be twice continuously differentiable with gradient denoted by $g(.)$ and Hessian by $G(.)$] imitate Newton's method, without requiring that the Hessian be available in explicit form. Instead, they compute approximations $\{B_i\}$, say, to the Hessian matrices at the various iterates $\{x_i\}$ which are generated. In standard quasi-Newton methods, the new approximation B_{i+1} is required to satisfy what is known as the *Secant* (or *quasi-Newton*) *Equation* (Dennis [2]):-

$$B_{i+1} s_i = y_i . \tag{1}$$

In this equation, s_i is the step taken from x_i to x_{i+1}:

$$s_i = x_{i+1} - x_i , \tag{2}$$

and y_i is the corresponding change in the gradient:

$$y_i = g(x_{i+1}) - g(x_i) . \tag{3}$$

Equation (1) is a particular approximation to the *Newton Equation*:-

$$G(x(\tau^*)) \left.\frac{dx}{d\tau}\right|_{\tau=\tau^*} = \left.\frac{dg}{d\tau}\right|_{\tau=\tau^*} \tag{4}$$

[obtained by applying the Chain Rule to the vector function $g(x(\tau))$, where $x(\tau)$ is any differentiable curve (denoted by $\mathbb{X}$, say) in $\mathbb{R}^n$]. Here, τ^* is

chosen so that $x(\tau^*) = x_{i+1}$, the most recent iterate. Equation (4) thus defines a condition which $G(x_{i+1})$ must satisfy and, by approximating the derivative $dg/d\tau$, we may obtain a relation which can then be used to construct the required estimate B_{i+1} of the Hessian $G(x_{i+1})$. For example, in the traditional quasi-Newton approach, the curve $\mathcal{X}$ is taken to be the straight line which interpolates the two most recent iterates:

$$x(\tau) \equiv x_i + \tau s_i . \tag{5}$$

By analogy, the behaviour of the gradient (when restricted to $\mathcal{X}$) is *approximated* by a linear polynomial interpolating the corresponding values of g, so that

$$\frac{dg}{d\tau} \approx g(x_{i+1}) - g(x_i) = y_i ,$$

and (1) is obtained as an approximation to (4).

The multi-step methods, developed by Ford and Moghrabi [7], considered polynomial forms (of degree larger than one) for $\mathcal{X}$. These polynomials are chosen to interpolate (for a prescribed set of values $\{\tau_k\}_{k=0}^{m}$) the m+1 most recent iterates:

$$x(\tau_k) = x_{i-m+k+1}, \text{ for } k = 0, 1, \ldots, m, \tag{6}$$

where, now, m is (in general) greater than one. The curve, thus, may be represented in its Lagrangian form:

$$x(\tau) \equiv \sum_{k=0}^{m} \mathcal{L}_k(\tau) x_{i-m+k+1}, \tag{7}$$

where $\mathcal{L}_k(\tau)$ is the standard Lagrangian polynomial

$$\mathcal{L}_k(\tau) \equiv \prod_{\substack{j=0 \\ j \neq k}}^{m} \{(\tau - \tau_j)/(\tau_k - \tau_j)\}. \tag{8}$$

By analogy, the gradient (when restricted to $\mathbb{X}$) may be approximated by the corresponding interpolatory polynomial

$$g(x(\tau)) \approx \sum_{k=0}^{m} \mathcal{L}_k(\tau) g(x_{i-m+k+1}). \tag{9}$$

We now determine $x'(\tau_m)$ from (8) and *estimate* $g'(\tau_m)$ by means of (9), so that we may substitute for the derivatives in (2) and obtain a condition for B_{i+1} to satisfy. Thus, we derive the requirement

$$B_{i+1} r_i = w_i, \tag{10}$$

where

$$r_i \stackrel{\Delta}{=} x'(\tau_m)$$

$$= \sum_{k=0}^{m} \mathcal{L}'_k(\tau_m) x_{i-m+k+1} \tag{11}$$

and

$$g'(x(\tau_m)) \approx \sum_{k=0}^{m} \mathcal{L}'_k(\tau_m) g(x_{i-m+k+1})$$

$$\stackrel{\Delta}{=} w_i. \tag{12}$$

The coefficients in (11) and (12) are given explicitly by

$$\mathcal{L}'_k(\tau_m) = (\tau_k - \tau_m)^{-1} \prod_{\substack{j=0 \\ j\neq k}}^{m-1} \{(\tau_m - \tau_j) / (\tau_k - \tau_j)\} \text{ for } k \neq m$$

and

$$\mathcal{L}'_m(\tau_m) = \sum_{j=0}^{m-1} (\tau_m - \tau_j)^{-1}.$$

It was shown by Ford and Moghrabi that r_i and w_i could be represented in terms of the most recent "step-vectors" $\{s_{i-j}\}_{j=0}^{m-1}$ and $\{y_{i-j}\}_{j=0}^{m-1}$:-

$$r_i = \sum_{j=0}^{m-1} s_{i-j} \{ \sum_{k=m-j}^{m} \mathcal{L}'_k(\tau_m) \}; \qquad (13a)$$

$$w_i = \sum_{j=0}^{m-1} y_{i-j} \{ \sum_{k=m-j}^{m} \mathcal{L}'_k(\tau_m) \}. \qquad (13b)$$

In the traditional quasi-Newton approach, m = 1 and the values $\{\tau_k\}_{k=0}^{1}$ are given by (using (5) and (6))

$$\tau_0 = 0; \qquad \tau_1 = 1.$$

Therefore, Ford and Moghrabi [7] chose to use a unit-spacing for the values $\{\tau_k\}_{k=0}^{m}$ required in the higher-degree interpolations discussed above:

$$\tau_k = k - m + 1, \text{ for } k = 0, 1, \ldots, m. \qquad (14)$$

Numerical experiments reported in that paper showed that, for m = 2 and m = 3, significant gains in computational efficiency can be made (by comparison with the standard approach for which m = 1), particularly as the dimension (n) of the problem increases. However, it was observed that (14) was not the only possible choice for the parameter-values and it is our intention here to consider some alternative choices. In the following section, we will introduce two different approaches to determining values for $\{\tau_k\}_{k=0}^{m}$ and, thus, the precise form of the polynomials which are used to interpolate the values $\{x_{i-m+k+1}\}_{k=0}^{m}$ and $\{g(x_{i-m+k+1})\}_{k=0}^{m}$.

2. Alternative Parameter Values

In describing the different approaches to defining the values of $\{\tau_k\}_{k=0}^{m}$, it will be useful to employ metrics on $\mathbb{R}^n$ of the following general form:-

$$\Phi_M(z_1, z_2) \triangleq [(z_1 - z_2)^T M(z_1 - z_2)]^{1/2} \qquad (15)$$

for all points z_1, $z_2 \in \mathbb{R}^n$ and for a given symmetric positive-definite matrix $M \in \mathbb{R}^{nxn}$.

(A) THE ACCUMULATIVE APPROACH

This approach locates the iterates $\{x_{i-m+k+1}\}_{k=0}^{m}$ on the curve by focussing attention on one of the points ($x_{i-m+j+1}$, say) as the chosen

"base-point" or "origin". This point corresponds to the value τ_j and we therefore select τ_j to be zero. For any integer k in the range [0, m], except k = j, the value τ_k (corresponding to the iterate $x_{i-m+k+1}$) is now computed by accumulating the distance (measured by the chosen metric Φ_M) between each consecutive pair of points in the sequence from $x_{i-m+j+1}$ to $x_{i-m+k+1}$, inclusive. In carrying out this computation, we impose the convention that iterates earlier than $x_{i-m+j+1}$ in the sequence are to correspond to negative values of τ, and points occurring later than $x_{i-m+j+1}$ to positive values. Thus, we have

$$\tau_k = -\sum_{p=k+1}^{j} \Phi_M(x_{i-m+p+1},\ x_{i-m+p}),\ \text{for } k < j, \qquad (16a)$$

$$\tau_j = 0, \qquad (16b)$$

$$\tau_k = \sum_{p=j+1}^{k} \Phi_M(x_{i-m+p+1},\ x_{i-m+p}),\ \text{for } k > j. \qquad (16c)$$

In our implementation, we have chosen j = m-1, thus continuing the (somewhat arbitrary) convention that $\tau_{m-1} = 0$, so that equations (16) become:-

$$\tau_{m-1} = 0, \quad \tau_m = \Phi_M(x_{i+1}, x_i), \qquad (17a)$$

$$\tau_k = -\sum_{p=k+1}^{m-1} \Phi_M(x_{i-m+p+1},\ x_{i-m+p}),\ \text{for } k = 0,\ 1, \ldots,\ m-2. \qquad (17b)$$

Equivalently,

$$\tau_m = \Phi_M(x_{i+1}, x_i),$$

$$\tau_k = \tau_{k+1} - \Phi_M(x_{i-m+k+2}, x_{i-m+k+1}), \text{ for } k = 0, 1, \ldots, m-2.$$

Evidently, as long as no consecutive pair of iterates is identical, the values $\{\tau_k\}_{k=0}^{m}$ yielded by this approach satisfy :-

$$\tau_i < \tau_{i+1}, \quad i = 0, 1, \ldots, m-1. \tag{18}$$

(B) THE FIXED-POINT APPROACH

As with the first approach, we fix attention upon one of the iterates ($x(\tau_j) \equiv x_{i-m+j+1}$, say). This time, however, we determine a value for τ_k ($k \in [0, m]$, $k \neq j$) by measuring (again using the chosen metric Φ_M) the distance from $x_{i-m+j+1}$ to $x_{i-m+k+1}$ *directly*. The same convention regarding positive and negative values of τ is also observed here. In this method, different forms of the algorithm result from distinct choices of the index j. (For example, for m = 2 and j = m-1 = 1, this construction is identical to the "Accumulative Approach" described previously.) To be specific, we have selected j = m, so that the "base-point" for the measurement of distances is the most recent iterate, x_{i+1}. It follows that

$$\tau_m = 0 \; ; \qquad \tau_k = -\Phi_M(x_{i+1}, x_{i+k-m+1}), \text{ for } k = 0, \ldots, m-1. \tag{19}$$

In this approach, it is possible that some of the parameters $\{\tau_k\}_{k=0}^{m}$ will be assigned equal (or nearly equal) values, so that some form of remedial action (for example, varying the metric or ignoring some of the iterates) will be necessary. In the tests reported below, we reverted to the unit-spaced method in such circumstances.

Clearly, for both the "Accumulative" and "Fixed-Point" approaches, different choices of the matrix M in the definition of the metric Φ_M (equation (15)) will lead to different values for the parameters $\{\tau_k\}_{k=0}^{m}$ and, thus, to differing algorithms. In the next section, we introduce some possible choices for M and discuss how the expressions thus arising for $\{\tau_k\}_{k=0}^{m}$ may be computed efficiently.

3. Particular Choices for the Metric

Ford and Moghrabi [7] reported that numerical results obtained from the "unit-spaced" methods were better (over the range of dimensions [2 to 80] considered in their experiments) for m = 2 than for m = 3. We conjecture that this may be due to the fact that, as m increases, the relative contribution of s_i to r_i and of y_i to w_i is reduced, so that the most recent step-vectors (namely, s_i and y_i) have a decreased effect on the formation of B_{i+1}. Thus, for m = 2, r_i is given (in normalised form) by

$$r_i = s_i - (1/3)s_{i-1},$$

while, for m = 3, it is given by

$$r_i = s_i - (7/11)s_{i-1} + (2/11)s_{i-2},$$

with corresponding expressions for w_i in terms of y_i, y_{i-1} and y_{i-2}. It is clear that the relative contribution of s_i and y_i has decreased. Whether similar features will occur in the methods being proposed here will depend, of course, on the metric Φ_M in use and the values for $\{\tau_k\}_{k=0}^{m}$ which are thus generated. Preliminary experiments indicated that, of the methods considered, those for which m = 2 were generally superior to those corresponding to m = 3. It was therefore concluded that the extra storage and housekeeping required by the "m = 3" methods was not worthwhile and we shall therefore, for the remainder of the paper, restrict our consideration to the case m = 2.

We now discuss some of the possible choices for the matrix M defining the metric Φ_M and consider the resulting algorithms.

(A) ACCUMULATIVE ALGORITHMS

Algorithm A1

This algorithm is generated by choosing M to be the unit matrix. Thus, from (15) and (17), we have

$$\tau_2 = \|s_i\|_2, \quad \tau_1 = 0 \quad \text{and} \quad \tau_0 = -\|s_{i-1}\|_2. \tag{20}$$

Algorithm A2

For this algorithm, we choose M to be the matrix B_i (the approximation to the Hessian at x_i). Hence, in addition to $\tau_1 = 0$, we have

$$\tau_0 = -[s_{i-1}^T B_i s_{i-1}]^{1/2}; \qquad \tau_2 = [s_i^T B_i s_i]^{1/2}.$$

If, as was the case with the algorithms tested here, x_{i+1} is determined from x_i by means of a line-search along the "quasi-Newton" step $-B_i^{-1} g(x_i)$, then $\exists t_i > 0$ such that

$$x_{i+1} = x_i - t_i B_i^{-1} g(x_i).$$

Therefore,

$$s_i = -t_i B_i^{-1} g(x_i)$$

and

$$\tau_2 = -[-t_i s_i^T g(x_i)]^{1/2}. \tag{21}$$

Thus, the term $-[s_i^T B_i s_i]^{1/2}$, requiring $O(n^2)$ operations, is replaced by the more efficient expression (necessitating O(n) operations) given in (21). In order to reduce the cost of calculating τ_0 (which would similarly require $O(n^2)$ operations), we resort to an approximation. Although the multi-step methods we are developing here do *not* satisfy the Secant Equation (1), it seems reasonable to assume that (1) will hold approximately for the matrices B_i generated by these methods. We claim that this assumption is valid because the intention is that B_i shall approximate $G(x_i)$, which *does* satisfy the Newton Equation (4), from which (1) was derived. Thus, applying (1) with i+1 replaced by i, we obtain

$$\tau_0 \approx -[s_{i-1}^T y_{i-1}]^{1/2},$$

and this quantity $(-[s_{i-1}^T y_{i-1}]^{1/2})$ is the value used for τ_0 used in the implementation of this method.

Algorithm A3

If we now choose M in (15) to be B_{i+1} and use similar techniques of approximation to those employed in constructing algorithm A2 above, we obtain:-

$$\tau_1 = 0; \qquad \tau_2 = [s_i^T B_{i+1} s_i]^{1/2} \approx [s_i^T y_i]^{1/2} \tag{22a}$$

and

$$\tau_0 = -[s_{i-1}^T B_{i+1} s_{i-1}]^{1/2} \approx -[s_{i-1}^T y_{i-1}]^{1/2}. \tag{22b}$$

(B) FIXED-POINT ALGORITHMS

Algorithm F1

Choosing M to be the unit matrix again, we have (from (15) and (19)), the following values for τ:-

$$\tau_2 = 0, \ \tau_1 = -\|s_i\|_2, \text{ and } \tau_0 = -\|s_i + s_{i-1}\|_2 \ . \tag{23}$$

Algorithm F2

For $M = B_i$, the resulting values are (using the arguments employed in constructing algorithm A3):-

$$\tau_2 = 0, \quad \tau_1 = -[-t_i s_i^T g(x_i)]^{1/2}, \tag{24a}$$

and

$$\tau_0 = -[-t_i s_i^T g(x_i) + 2 s_i^T B_i s_{i-1} + s_{i-1}^T B_i s_{i-1}]^{1/2}$$

$$\approx -[-t_i s_i^T g(x_i) + 2 s_i^T y_{i-1} + s_{i-1}^T y_{i-1}]^{1/2} . \tag{24b}$$

We remark that it is simple to construct variations of this particular algorithm; for example, the term $2 s_i^T y_{i-1}$ in the expression for τ_0 could (with equal validity) be replaced with $-2 t_i s_{i-1}^T g(x_i)$ (by rewriting the term $2 s_i^T B_i s_{i-1}$ in the form $2 s_{i-1}^T B_i s_i$) or with $[s_i^T y_{i-1} - t_i s_{i-1}^T g(x_i)]$ (through rewriting the same expression as $[s_i^T B_i s_{i-1} + s_{i-1}^T B_i s_i]$).

Algorithm F3

Finally, using $M = B_{i+1}$, we obtain:-

$$\tau_2 = 0, \quad \tau_1 \approx -(s_i^T y_i)^{1/2}, \tag{25a}$$

and

$$\tau_0 \approx -[s_i^T y_i + 2 s_{i-1}^T y_i + s_{i-1}^T y_{i-1}]^{1/2} . \tag{25b}$$

Again, it is clearly possible (as with F2) to construct variations of this method.

4. Numerical experiments

The new methods described above, together with the standard BFGS method and the unit-spaced 2-step method (denoted by the name M2) introduced in Ford & Moghrabi [7], were applied to the minimization of a test set of 107 functions. This set included many test functions which are well-known and well-documented in the literature, such as the extended Powell singular function, the Chained Wood function, Coope's "F55" function, the Trigonometric function, the Engvall function and the Gragg-Levy function. Some of the other functions in the test set are described below. Each function was minimized from four different starting points, giving a total of 428 test problems. The dimensions of these functions ranged from 2 to 80. All the multi-step methods were implemented using the BFGS formula, with s_i and y_i substituted by the appropriate definitions of r_i and w_i (see equations (13a) and (13b)). In all cases, x_{i+1} was calculated from x_i by means of a line-search algorithm which accepted the predicted point if the two conditions given below were satisfied and which, otherwise, used step-doubling and cubic interpolation, as necessary. For x_{i+1} to be accepted, the following conditions were imposed (see Fletcher [3]):-

$$f(x_{i+1}) \leq f(x_i) + 10^{-2} s_i^T g(x_i) \ ;$$

$$s_i^T g(x_{i+1}) > s_i^T g(x_i) .$$

Where the dimension of the function was 10 or greater, the initial Hessian approximation (the unit matrix) was scaled by the method of Shanno & Phua [10] before the first update was performed.

Limitations of space preclude giving a full report of the results obtained from these experiments. We will, therefore, first present a brief summary and discussion of the performance (on the full problem set) of all the methods and then give more details for a subset of ten functions and a selected group of four of the methods. Over the complete set of 428 problems, the eight methods under consideration yielded the following results (the term "evaluation" refers to one evaluation of the relevant function and its gradient; all times are measured in seconds and percentages relate to the corresponding figure for the BFGS method):-

Table 1 : Overall Results for 428 Problems

Method	Total Evaluations	Total Iterations	Total Time	No. of Failures
BFGS	98425 (100%)	91304 (100%)	3402.16 (100%)	0
M2	90275 (91.7%)	82153 (90.0%)	3024.58 (88.9%)	0
A1	80616 (81.9%)	73435 (80.4%)	2660.09 (78.2%)	0
A2	83752 (85.1%)	75676 (82.9%)	2709.01 (79.6%)	1
A3	82318 (83.6%)	72648 (79.6%)	2703.47 (79.5%)	0
F1	79390 (80.7%)	71375 (78.2%)	2620.62 (77.0%)	0
F2	79276 (80.5%)	71743 (78.6%)	2418.42 (71.1%)	0
F3	82895 (84.2%)	72620 (79.5%)	2657.04 (78.1%)	0

We observe that all six of the methods proposed here show a substantial improvement (in terms of evaluations, iterations and time) over the standard BFGS method and over the unit-spaced 2-step method M2. In particular, F2 requires 29% *less* than the execution time needed by the BFGS method and the other five new methods show improvements which are not much lower. The improvement in the number of function evaluations required is not quite so high (about 20% for F2). The main reason for this appears to be that the new methods show an increasing improvement over BFGS as the dimension (and therefore, in general, the time required to solve the problem) increases, which we regard as a very desirable feature. In order to substantiate this assertion, we show, in Tables 2 to 4, a breakdown of the results shown in Table 1. We have classified the test problems into three subsets, according to the dimension of the problem. We observe that, for the low-dimensional problems (Table 2), the improvement in performance (if it exists) is not very large (although even in this case, a more detailed breakdown demonstrates that the performance of the multi-step methods, by comparison with that of BFGS, tends to improve as the dimension increases). However, as we move to higher dimensions (Tables 3 and 4), the increasing benefits yielded by the new methods are clear. We also draw attention to the fact that, in Table 4, the improvements in time correspond closely to the improvements in the number of evaluations and in the number of iterations. This provides evidence of a practical nature that the overheads of implementing the new methods become increasingly negligible (by comparison with the costs of evaluating the function and its gradient, and with housekeeping costs such as updating the matrix) as the dimension rises.

Table 2 : Results for Dimensions from 2 to 15 (116 Problems)

Method	Total Evaluations	Total Iterations	Total Time	No. of Failures
BFGS	20619 (100%)	16865 (100%)	51.11 (100%)	0
M2	20940 (101.6%)	16733 (99.2%)	52.43 (102.6%)	0
A1	20164 (97.8%)	16328 (96.8%)	50.38 (98.6%)	0
A2	19677 (95.4%)	15606 (92.5%)	46.48 (90.9%)	0
A3	20431 (99.1%)	14897 (88.3%)	47.17 (92.3%)	0
F1	18852 (91.4%)	15016 (89.0%)	48.90 (95.7%)	0
F2	21161 (102.6%)	17446 (103.4%)	50.23 (98.3%)	0
F3	21254 (103.1%)	15669 (92.9%)	51.01 (99.8%)	0

Table 3 : Results for Dimensions from 16 to 45 (172 Problems)

Method	Total Evaluations	Total Iterations	Total Time	No. of Failures
BFGS	43009 (100%)	40152 (100%)	737.88 (100%)	0
M2	38965 (90.6%)	35771 (89.1%)	672.07 (91.1%)	0
A1	33956 (79.0%)	31302 (78.0%)	587.67 (79.6%)	0
A2	37109 (86.3%)	33708 (84.0%)	611.27 (82.8%)	0
A3	34738 (80.8%)	31275 (77.9%)	588.72 (79.8%)	0
F1	34254 (79.6%)	31052 (77.3%)	586.28 (79.5%)	0
F2	34161 (79.4%)	31132 (77.5%)	549.70 (74.5%)	0
F3	34941 (81.2%)	31148 (77.6%)	585.29 (79.3%)	0

Table 4 : Results for Dimensions from 46 to 80 (140 Problems)

Method	Total Evaluations	Total Iterations	Total Time	No. of Failures
BFGS	34797 (100%)	34287 (100%)	2613.17 (100%)	0
M2	30370 (87.3%)	29649 (86.5%)	2300.08 (88.0%)	0
A1	26496 (76.1%)	25805 (75.3%)	2022.04 (77.4%)	0
A2	26966 (77.5%)	26362 (76.9%)	2051.26 (78.5%)	1
A3	27149 (78.0%)	26476 (77.2%)	2067.58 (79.1%)	0
F1	26284 (75.5%)	25307 (73.8%)	1985.44 (76.0%)	0
F2	23954 (68.8%)	23165 (67.6%)	1818.49 (69.6%)	0
F3	26700 (76.7%)	25803 (75.3%)	2020.74 (77.3%)	0

On the basis of the overall results presented above, we selected A1 (as the best "Accumulative" method) and F2 (as the best "Fixed-Point" method), in addition to the BFGS method and M2, for the purpose of exhibiting a more detailed comparison of relative behaviour. The chosen subset of functions, together with their dimensions and starting-points, is described in Table 5 (unless otherwise stated, a full definition of each function may be found in Moré, Garbow & Hillstrom [8], except for the "Quadratic" function, which is defined below). The notation $[\alpha, \beta, \ldots, \omega]^*$ is to be understood as specifying that the vector $[\alpha, \beta, \ldots, \omega]$ is repeated as many times as necessary to create a vector of the required dimension. (The "Quadratic" function is defined by

$$f(x) \equiv \frac{1}{2} x^T L L^T x,$$

where L is the unit-lower-triangular matrix

$$L_{ij} = 1/(i - j + 1), \text{ for } i \geq j.)$$

The results of executing the four algorithms (BFGS, M2, F2 and A1) on this sample set of 40 problems are given in Table 6. Each entry consists of two integers, the first of which is the number of function/gradient evaluations required to minimize the function from the given starting-point, while the second (in brackets) gives the number of iterations. The best performance on each problem (judged, as is conventionally done, by the number of evaluations) is indicated by the symbol "‡". The total number of best performances for each algorithm is given in the final row of the Table ("SCORES"), together with the totals for evaluations and iterations in the penultimate row.

These results are illustrative of those obtained from applying the four methods to the full test set (although it would be unwise, of course, to draw firm conclusions about the relative virtues of methods from such a small sample). They underline the advantages accruing from use of the new methods and show clearly how the "Fixed-Point" algorithm F2, in particular, is increasingly superior to the standard BFGS method and to the unit-spaced method (M2) as the dimension of the problem rises. On the sixteen problems with dimension 50 or greater, F2 produces the best performance of the four methods on no fewer than fourteen occasions. Improvements (in the number of function evaluations) of 20-30% and greater over the BFGS method are common for such dimensions.

5. Summary and Conclusions

A number of multi-step algorithms have been developed which rely on one of two approaches to defining the distribution of recent iterates on an interpolating curve. It has been shown how the quantities required to implement such algorithms may be efficiently computed. (We observe, at this point, that the important issue of preserving positive-definiteness in the Hessian approximations generated by multi-step methods is covered by the results of Ford and Moghrabi [7].) The new algorithms have been experimentally compared with the standard BFGS method and with the unit-spaced 2-step method developed earlier by Ford and Moghrabi. The numerical results strongly support use of the new methods introduced here. One particularly encouraging feature of their performance is the increasing gain in efficiency as the dimension of the problem rises. The improvements observed in the numerical experiments imply that the extra storage and computation required [both of which are O(n)] in implementing these methods are more than justified.

Further work currently in progress seeks to combine the techniques of Ford and Ghandhari [5,6,7] with the approaches considered here in order to exploit the function-values which are available at each iteration. Also under consideration is the construction of "hybrid" methods, where the selection of the algorithm to be used depends on the dimension of the function. Finally, it would be of interest to investigate limited-memory implementations of these methods.

6. Acknowledgment

One of us (I.A.M.) gratefully acknowledges the support of the Hariri Foundation during this research.

References

[1] A.R. Conn, N.I.M. Gould and Ph.L. Toint, Testing a class of methods for solving minimization problems with simple bounds on the variables, Research Report CS-86-45, University of Waterloo (1986).

[2] J.E. Dennis, On some methods based on Broyden's secant approximation to the Hessian; in *Numerical Methods for Non-linear Optimization*, ed. F. Lootsma (Academic Press, London, 1972).

[3] R. Fletcher, *Practical Methods of Optimization* (second edition) (Wiley, New York, 1987).

[4] J.A. Ford and R.-A. Ghandhari, On the use of function-values in unconstrained optimisation, *J. Comput. Appl. Math.* **28** (1989) 187 - 198.

[5] J.A. Ford and R.-A. Ghandhari, Efficient utilisation of function-values in quasi-Newton methods, in: *Colloquia Mathematica Societatis János Bolyai* **59** (Numerical Methods, Miskolc, 1990), ed. D. Greenspan and P. Rózsa, (North Holland, Amsterdam, 1991).

[6] J.A. Ford and R.-A. Ghandhari, On the use of curvature estimates in quasi-Newton methods, *J. Comput. Appl. Math.* **35** (1991) 185 - 196.

[7] J.A. Ford and I.A. Moghrabi, Multi-step quasi-Newton methods for optimization, Department of Computer Science Technical Report CSM-171, University of Essex, 1992.

[8] J.J. Moré, B.S. Garbow and K.E. Hillstrom, Testing unconstrained optimization software, *TOMS* **7** (1981) 17 - 41.

[9] D.F. Shanno and K.H. Phua, Algorithm 500: Minimization of unconstrained multivariate functions, *TOMS* **2** (1976) 87 - 94.

[10] Ph.L. Toint, On large scale nonlinear least squares calculations, *SIAM J. Sci. Stat. Comput.* **8** (1987) 416 - 435.

J.A. Ford

Department of Computer Science

University of Essex

Wivenhoe Park

Colchester

Essex

United Kingdom

CO4 3SQ

Fax : 010.44.206.872788

e-mail : fordj@essex.ac.uk (janet)

TABLE 5

Problem name	Dimension	Starting-points	
Rosenbrock	2	[a]	(-1.2, 1.0)
		[b]	(-120, 100)
		[c]	(20, -20)
		[d]	(6.39, -0.221)
Chebyquad	5	[a]	(0.2, 0.4, 0.6, 0.8, 1.0)
		[b]	(0, 2, 3, 4, 5)
		[c]	(2, -1, 0, 1, 2)
		[d]	(0.0625, 0.125, 0.25, 0.5, 1.0)
Penalty I	10	[a]	(1, 2, 3,..., 10)
		[b]	([5, -5]*)
		[c]	([2, 1, 0, -1, -2]*)
		[d]	(-10, -20, -30,..., -100)
Variably-dimensioned	20	[a]	(0.95, 0.9, 0.85,...,0.0)
		[b]	([10, 5, 0, -5, -10]*)
		[c]	(5, 10, 15,..., 100)
		[d]	([-100, 75, -50, 25]*)
VAR (Conn et al [1])	30	[a]	(1, 1, 1,...,1)
		[b]	([-2, -1, 0, 1, 2]*)
		[c]	([-5, -3, -1]*)
		[d]	([-1.5, -1.4, -1.3,...,-0.1]*)
Extended Rosenbrock	40	[a]	([-1.2, 1.0]*)
		[b]	([-120, 100]*)
		[c]	([1, -2, 3, -4,...,9, -10]*)
		[d]	(20, 20,..., 20)
Toint Merged Quadratic (Toint [10])	50	[a]	(5, 5, 5,..., 5)
		[b]	([1, -2, 3, -4, 5]*)
		[c]	([-1, 2]*)
		[d]	([-10, -9, -8,.., -1]*)
Discrete Boundary-Value	60	[a]	([1, 2, 3,..., 10]*)
		[b]	([-2, -1, 0, 1, 2]*)
		[c]	([10, 0, -10]*)
		[d]	([10, -9, 8, -7,..., -1]*)
Discrete Integral Equation	70	[a]	([3, 2, 1, 0,-1,-2,-3]*)
		[b]	([5,-4, 3,-2, 1,-1, 2,...,-5]*)
		[c]	([7, 6, 5,..., 1,-7,-6,..,-1]*)
		[d]	(10, 10, 10,..., 10)
Quadratic	80	[a]	([1, 2, 3, 4, 5, 5,..., 2, 1]*)
		[b]	([-1, 1, -2, 2,..., -5, 5]*)
		[c]	([20, 19, 18,..., 2, 1]*)
		[d]	([100, 10, -10, -100]*)

TABLE 6

Problem		BFGS	M2	F2	A1
Rosenbrock	[a]	45(35)‡	51(46)	54(45)	51(37)
	[b]	544(416)‡	610(482)	638(510)	559(436)
	[c]	161(127)‡	195(151)	199(162)	176(148)
	[d]	73(59)‡	77(64)	87(72)	78(68)
Chebyquad	[a]	30(18)‡	35(19)	31(19)	31(17)
	[b]	157(151)	129(91)‡	156(119)	146(89)
	[c]	131(107)	113(66)	69(52)‡	111(76)
	[d]	33(23)‡	38(25)	38(27)	34(23)
Penalty I	[a]	86(74)	95(77)	72(61)‡	74(65)
	[b]	145(118)	148(131)	139(122)‡	141(119)
	[c]	133(107)	140(117)	148(119)	123(101)‡
	[d]	221(178)	232(189)	246(201)	216(169)‡
Variably-Dimensioned	[a]	26(25)	21(20)	19(18)	18(16)‡
	[b]	62(61)	60(57)	39(38)‡	42(41)
	[c]	97(96)	91(87)	58(57)‡	67(65)
	[d]	92(91)	90(88)	68(66)‡	75(74)
VAR	[a]	38(37)	35(34)	31(30)‡	33(32)
	[b]	66(65)	59(58)	60(59)	56(55)‡
	[c]	70(69)	65(64)	62(61)‡	65(64)
	[d]	62(61)	54(53)‡	58(57)	54(53)‡
Extended Rosenbrock	[a]	47(39)‡	53(48)	49(44)	53(45)
	[b]	207(177)‡	228(192)	220(193)	207(183)
	[c]	262(252)	239(225)	252(233)	205(193)‡
	[d]	118(105)‡	129(116)	142(125)	124(112)
Toint Merged Quadratic	[a]	333(332)	306(305)	178(177)‡	235(234)
	[b]	243(242)	215(214)	141(139)‡	165(162)
	[c]	102(101)	90(89)	79(77)‡	89(87)
	[d]	476(473)	396(395)	252(250)‡	331(321)
Discrete Boundary-Value	[a]	240(239)	201(200)	188(187)‡	193(192)
	[b]	227(226)	192(191)	163(162)‡	168(167)
	[c]	254(253)	229(228)	212(211)	208(207)‡
	[d]	273(272)	238(237)	179(178)‡	221(220)
Discrete Integral Equation	[a]	46(45)	45(44)	37(36)‡	41(40)
	[b]	69(68)	69(68)	55(54)‡	63(62)
	[c]	96(95)	88(87)	77(76)‡	83(82)
	[d]	149(148)	131(130)	108(107)‡	118(117)
Quadratic	[a]	120(119)	101(100)	95(94)‡	99(98)
	[b]	62(61)	52(51)	56(55)	49(48)‡
	[c]	165(164)	137(136)	113(112)‡	124(123)
	[d]	131(130)	106(105)	100(99)‡	100(99)‡
TOTALS		5892(5459)	5583(5080)	4968(4504)	5026(4540)
SCORES (‡)		9	2	22	9