Performance of a multifrontal scheme for partially separable optimization

(Revised)

by A.R. Conn[1], N.I.M. Gould[2], M. Lescrenier[3] and Ph.L. Toint[4]

Report 88/4

January 11, 1989

Abstract. We consider the solution of partially separable minimization problems subject to simple bounds constraints. At each iteration, a quadratic model is used to approximate the objective function within a trust region. To minimize this model, the iterative method of conjugate gradients has usually been used. The aim of this paper is to compare the performance of a direct method, a multifrontal scheme, with the conjugate gradient method (with and without preconditioning). To assess our conclusions, a set of numerical experiments, including large dimensional problems, is presented.

[1] Department of Combinatorics and Optimization, University of Waterloo, Ontario, Canada.
[2] Computer Science and Systems Division, Harwell Laboratory, Oxfordshire, England.
[3] Belgian National Fund for Scientific Research, F.U.N.D.P. Namur, Belgium.
[4] Department of Mathematics, F.U.N.D.P. Namur, Belgium.

Keywords : partially separable optimization, multifrontal Gaussian elimination.

This report is issued simultaneously at Harwell Laboratory, England and the Department of Combinatorics and Optimization, University of Waterloo, Ontario, Canada.

Performance of a multifrontal scheme for partially separable optimization

by A.R. Conn[1], N.I.M. Gould[2], M. Lescrenier[3]
and Ph.L. Toint[4]

January 11, 1989

Abstract. We consider the solution of partially separable minimization problems subject to simple bounds constraints. At each iteration, a quadratic model is used to approximate the objective function within a trust region. To minimize this model, the iterative method of conjugate gradients has usually been used. The aim of this paper is to compare the performance of a direct method, a multifrontal scheme, with the conjugate gradient method (with and without preconditioning). To assess our conclusions, a set of numerical experiments, including large dimensional problems, is presented.

[1] Department of Combinatorics and Optimization, University of Waterloo, Ontario, Canada.
[2] Computer Science and Systems Division, Harwell Laboratory, Oxfordshire, England.
[3] Belgian National Fund for Scientific Research, F.U.N.D.P. Namur, Belgium.
[4] Department of Mathematics, F.U.N.D.P. Namur, Belgium.

Keywords : partially separable optimization, multifrontal Gaussian elimination.

This report is issued simultaneously at Harwell Laboratory, England and the Department of Combinatorics and Optimization, University of Waterloo, Ontario, Canada.

1 Introduction.

This paper is concerned with the solution of partially separable optimization problems (defined in Section 2). Such problems appear in a large majority of nonlinear minimization applications, for example finite-elements, network problems and others. The formalism was first introduced by Griewank and Toint [8] and methods using this particular structure have proved to be extremely successful for large dimensional problems (see [9] for instance).

To solve these problems, a trust region type algorithm may be applied, which requires the partial solution of a quadratic minimization problem at each step of an iterative scheme. Up to now, only iterative methods, specifically (preconditioned and truncated) conjugate gradient schemes, have been used in practice to solve the quadratic minimization problem. The aim of this paper is to test the use of direct methods, particularly multifrontal schemes, for this purpose.

The authors are aware that this type of direct method can only be used when the solution of the quadratic problem can be found by solving a linear system, that is, when the quadratic has a finite solution. Such a situation normally arises when the Hessian matrix of the quadratic is positive definite. In the indefinite case, the authors use a simple strategy which consists of computing directions of negative curvature. However, they realize that a more sophisticated strategy like the Levenberg-Marquardt algorithm (see [11]) or an attempt to solve the trust region problem by another method may be more appropriate.

The paper is organized as follows. Section 2 defines the concept of partial separability. Section 3 describes the trust region algorithm used to solve partially separable problems. Iterative and direct methods to solve the quadratic minimization problem are proposed in Sections 4 and 5 respectively. In Section 6 we discuss the numerical experiments and conclusions are drawn in Section 7. We end with an appendix that describes some additional test functions not available in the literature.

2 Partial separability.

We consider the simple bound constrained minimization problem

$$\min \quad f(x) \tag{1}$$

subject to the bounds

$$a_j \leq x_j \leq b_j \tag{2}$$

where x is a vector of $\mathbf{R}^n$ and f is a so called *partially separable function*, that is a function of the form

$$f(x) = \sum_{i=1}^{m} f_i(x) \tag{3}$$

where the *element functions* $f_i(x)$ have Hessian matrices of low rank compared with n, the dimension of the problem.

A typical case is when each element function only depends on a small subset of the variables called the *elemental variables* of that element. It is also frequently the case that, for some elements, the number of elemental variables can be further reduced by applying, for each one of these elements, a linear transformation of its elemental variables in its *internal variables.* For every element, that is for $i = 1, \ldots, m$, the complete transformation from the original variables of the problem (the vector x) to elemental or (when applicable) to internal variables is then given by

$$y_i = U_i x, \tag{4}$$

where the matrix U_i has fewer than n rows. For instance, given the partially separable function ($n = 3$)

$$f(x) = x_1^2 + (x_1 - x_2)^2 + (x_2 - x_3)^2, \tag{5}$$

we have that

$$f_1(x) = x_1^2, \quad f_2(x) = (x_1 - x_2)^2 \text{ and } f_3(x) = (x_2 - x_3)^2, \tag{6}$$

where the sets of elemental variables corresponding to the elements are given by $\{x_1\}$, $\{x_1, x_2\}$ and $\{x_2, x_3\}$. These sets can be further reduced for elements 2 and 3 by defining two 1-component vectors y_2 and y_3 of internal variables by

$$y_2 = \begin{pmatrix} 1 & -1 & 0 \end{pmatrix} x \text{ and } y_3 = \begin{pmatrix} 0 & 1 & -1 \end{pmatrix} x. \tag{7}$$

For the first element, the elemental and internal variables coincide, and we have that

$$y_1 = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} x. \tag{8}$$

Equations (7) and (8) in our example correspond to (4), and hence

$$U_1 = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \quad U_2 = \begin{pmatrix} 1 & -1 & 0 \end{pmatrix} \text{ and } U_3 = \begin{pmatrix} 0 & 1 & -1 \end{pmatrix}. \tag{9}$$

The adaptability of partially separable methods to large problems comes mainly from a compact storage scheme for the Hessian approximation and the corresponding updating technique. The change of variables (4) allows us to consider new element functions $\hat{f}_i$ such that

$$f_i(x) = \hat{f}_i(y_i). \tag{10}$$

The gradients of f_i and $\hat{f}_i$ satisfy

$$g_i(x) = U_i^T \hat{g}_i(y_i), \tag{11}$$

while the Hessian approximations at x satisfy

$$H_i(x) = U_i^T \hat{H}_i(y_i) U_i. \tag{12}$$

The so called *partitioned updating technique* consists in storing and accessing only the gradients and Hessian approximations in internal variables, that is, the $\hat{g}_i(y_i)$ and $\hat{H}_i(y_i)$ from (11) and (12) respectively. The advantage clearly comes from the fact that the number of internal variables is much smaller than the total dimension of the problem and that the matrix approximating the Hessian of f

$$H(x) = \sum_{i=1}^{m} H_i(x) \tag{13}$$

is never explicitly assembled.

Finally, only the non empty columns of U_i (and pointers to the variables relevant to each column) are needed. For many practical problems, sets of the U_i differ *only* in the elemental variables that they select, see for instance U_2 and U_3 in (9). We would not therefore envisage storing each U_i, merely a set of variable-to-column pointers and require that commonly occuring operations, such as the products

$$u = U_i v \text{ and } u = U_i^T v \tag{14}$$

be performed by a user provided subroutine.

3 A trust region algorithm for partially separable problems.

The algorithm we propose for solving problem (1)–(2) is of trust region type and belongs to the class of methods described by Conn, Gould and Toint in [3].

At each iteration, we suppose that we have a feasible point $x^{(k)}$, the gradient $g^{(k)}$ of the objective function (3) at this point and a suitable symmetric approximation $H^{(k)}$ to the Hessian of f at $x^{(k)}$. In the remainder of this paper, by "Hessian" we will always mean an approximation of the true Hessian. This approximation may be computed by a secant updating formula such as the Broyden-Fletcher-Goldfarb-Shanno or the symmetric rank-one update for instance (see [5]). The gradient and Hessian of the partially separable function f are stored as described in Section 2. We approximate the objective function by a quadratic model

$$m^{(k)}(x^{(k)} + s) = f(x^{(k)}) + s^T g^{(k)} + \frac{1}{2} s^T H^{(k)} s, \tag{15}$$

in a region surrounding the current iterate defined by its radius $\Delta^{(k)}$. This region, called the trust region, is of the form

$$\|x - x^{(k)}\| \le \Delta^{(k)}. \tag{16}$$

It is convenient to choose the infinity norm, for then the shape of the trust region is aligned with the simple bounds of the problem (1)–(2). If we define

$$l_i^{(k)} = \max[a_i, x_i^{(k)} - \Delta^{(k)}], \quad u_i^{(k)} = \min[b_i, x_i^{(k)} + \Delta^{(k)}], \tag{17}$$

a trial point $x^{(k)} + s^{(k)}$ is constructed by finding an approximation to the solution of the trust region problem

$$\min\ m^{(k)}(x^{(k)} + s) \tag{18}$$

subject to the bounds

$$l_i^{(k)} \le x_i^{(k)} + s_i \le u_i^{(k)} \tag{19}$$

If the function value calculated at this new point is well approximated by its predicted value (the one given by the model), the point is accepted as the next iterate and the trust region is possibly enlarged; otherwise, the point is rejected and the trust region size decreased. More precisely, we compute the ratio of the achieved to the predicted reduction of the objective function,

$$\rho^{(k)} = \frac{f(x^{(k)}) - f(x^{(k)} + s^{(k)})}{f(x^{(k)}) - m^{(k)}(x^{(k)} + s^{(k)})} \tag{20}$$

and set

$$x^{(k+1)} = \begin{cases} x^{(k)} + s^{(k)} & \text{if } \rho^{(k)} > \mu, \\ x^{(k)} & \text{if } \rho^{(k)} \le \mu, \end{cases} \tag{21}$$

and

$$\Delta^{(k+1)} = \begin{cases} \gamma_0 \Delta^{(k)} & \text{if } \rho^{(k)} \le \mu, \\ \Delta^{(k)} & \text{if } \mu < \rho^{(k)} < \eta, \\ \gamma_2 \Delta^{(k)} & \text{if } \rho^{(k)} \ge \eta, \end{cases} \tag{22}$$

where $\gamma_0 < 1 < \gamma_2$, μ and η are appropriate numbers.

It remains now to describe our approximate solution of (18)–(19). We first compute the Generalized Cauchy Point $x_C^{(k)}$, which is defined as the first local minimizer of the univariate function

$$m^{(k)}(P[x^{(k)} - tg^{(k)}]) \tag{23}$$

with respect to $t \in \mathbf{R}$, where $P[\cdot]$ is the projection operator computed componentwise as

$$(P[x])_j = \begin{cases} l_j^{(k)} & \text{if } x_j \le l_j^{(k)}, \\ u_j^{(k)} & \text{if } x_j \ge u_j^{(k)}, \\ x_j & \text{otherwise.} \end{cases} \tag{24}$$

The Generalized Cauchy Point (GCP) is therefore the first local minimizer of the model, along the piecewise linear arc defined by projecting the steepest descent direction into the feasible domain of problem (1)–(2). We then choose $x^{(k)} + s^{(k)} = x_C^{(k)}$ if the reduced model's gradient at this point is smaller in norm than a fraction of the norm of the reduced model's gradient at $x^{(k)}$, that is if

$$\|Z(x_C^{(k)})^T \nabla m^{(k)}(x_C^{(k)})\| \le \eta^{(k)} \stackrel{\text{def}}{=} \min\left[0.1, \sqrt{\|Z(x^{(k)})^T g^{(k)}\|}\right] \|Z(x^{(k)})^T g^{(k)}\|, \tag{25}$$

where $Z(x)^T$ is the orthogonal projector onto the linear subspace corresponding to variables that are not at their bound at x. If the test (25) fails, we define $I(x_C^{(k)}, l^{(k)}, u^{(k)})$ as the active set of $x_C^{(k)}$ with respect to the bounds $l^{(k)}$ and $u^{(k)}$, that is the set of indices of the variables at the GCP violating or lying on a constraint of (19). We also define $C(x_C^{(k)})$ as the linear subspace of the variables that are free at the GCP, that is

$$C(x_C^{(k)}) = \text{span}\{e_i \mid i \notin I(x_C^{(k)}, l^{(k)}, u^{(k)})\} \tag{26}$$

where e_i is the i-th vector of the canonical basis of $\mathbf{R}^n$. In order to find $x^{(k)} + s^{(k)}$, we then compute a better approximation than the GCP to the solution of sub-problem (18)–(19) with the restriction that $s \in C(x_C^{(k)})$, so that the variables in the active set remain fixed at their value at the GCP. This gives us the trial point and can be calculated using an iterative or a direct method. This is the subject of the next two Sections.

4 Minimization of the quadratic model by iterative methods.

If the Hessian $H^{(k)}$, restricted to the subspace $C(x_C^{(k)})$, is positive definite and the bounds of sub-problem (18)–(19) that are inactive at the Cauchy point remain inactive, the solution of (18)–(19) may be obtained as the solution of a system of linear equations. However, this may be prohibitively expensive in the context of large scale optimization unless care is taken to solve the resulting linear system as efficiently as possible. To date, iterative schemes (in particular, truncated, preconditioned conjugate gradient methods) seem to have been the most popular approach for approximately solving the sub-problem, as reflected in the works of Toint [14], Steihaug [12], or Stoer [13].

Two main reasons explain this interest. The first one is that a truncated strategy that asymptotically takes the exact quasi-Newton step can save a significant amount of computation during the early iterations, when we are still far from the optimum. The second reason, and probably the main one, is that these types of methods do not involve operations on the matrix entries but only require matrix-vector products. This calculation, which represents the major cost of the algorithm, can be efficiently performed when the sparsity pattern of the matrix is taken into account. In the context of partially separable optimization, the matrix is the Hessian of the objective function, and the matrix-vector product does not require the assembly of the Hessian since it can be computed as

$$(\sum_{i=1}^{m} H_i)d = \sum_{i=1}^{m}(U_i^T \hat{H}_i U_i)d. \tag{27}$$

The conjugate gradient algorithm is applied, starting from $x = x_C^{(k)}$, to the sub-problem (18)–(19) with the restriction that the variables in the set $I(x_C^{(k)}, l^{(k)}, u^{(k)})$ are kept fixed throughout the process. Our algorithm optionally uses a preconditioner given by the Moore-Penrose pseudo-inverse of the diagonal matrix whose entries are the diagonal entries of the restricted Hessian. The algorithm is terminated at the point $\bar{x}$ if

1. the norm of the restricted gradient of the model, that is $||Z(x)^T \nabla m^{(k)}(x)||$, is less than $\eta^{(k)}$, for some $\eta^{(k)}$;

2. one or more of the unrestricted variables violate one of the bounds (19). The point $\bar{x}$ is then the point at which the first offending bound(s) is (are) encountered;

3. a direction of negative curvature is found, in which case $\bar{x}$ is chosen as the last point along this direction that still satisfies the bounds.

We refer the reader to [4] for a detailed description of this procedure.

A superlinear rate of convergence can be assured provided that the ratio of the norm of the restricted model's gradient at the final point to that at $x^{(k)}$ tends to zero as the iterates approach a Kuhn-Tucker point for the problem and the matrices $H^{(k)}$ restricted to the set of variables active at the solution satisfy the Dennis-Moré condition (see [5]). A suitable choice for $\eta^{(k)}$ in order to satisfy the first condition is given by the definition of $\eta^{(k)}$ in (25).

5 Minimization of the quadratic model by a direct method.

Let us now consider the solution by a direct method of the sub-problem (18)–(19) with the additional constraint that variables that are at their bound at the GCP remain fixed.

We already remarked that the advantage of using the conjugate gradient method to solve (18)–(19) is that there is no need to assemble the Hessian explicitly. Surprisingly, this advantage can be maintained for a class of direct methods called the frontal methods. These methods solve symmetric positive-definite systems of linear equations

$$Ax = b \tag{28}$$

by Gaussian elimination, where the matrix A has a finite-element representation

$$A = \sum_r B^{(r)}, \tag{29}$$

and where each $B^{(r)}$ is zero except in a small number of rows and columns.

In frontal methods, advantage is taken of the fact that elimination steps

$$a_{ij} := a_{ij} - \frac{a_{i\ell}a_{\ell j}}{a_{\ell\ell}} \tag{30}$$

do not have to wait for all the assembly steps

$$a_{ij} := a_{ij} + b_{ij}^{(r)} \tag{31}$$

from (29) to be complete. The operation (30) can be performed as soon as the pivot row (and column) is fully summed, that is, as soon as all the operations (31) have been completed for them. The fully summed rows and columns (not yet eliminated) are stored in a so-called frontal matrix whose size can be maintained sufficiently small.

Frontal methods are particularly well suited to our framework, because of the obvious similarity between (29) and (13). The class of frontal methods includes several algorithms for varying structure in the matrix A (for instance, band, arrowhead, ...). As we were interested in a general purpose solver, we chose to use the approach introduced by Irons [10], allowing for general sparsity pattern of A. We realize however that using this method for simple sparsity patterns may be overly complicated, and simpler direct methods may be more appropriate. We also note that this technique can also exploit sparsity by using several distinct frontal matrices: we will thus refer to it as the "multifrontal" method.

However, this scheme does not apply to indefinite matrices. To overcome this difficulty, Duff and Reid [7] followed Bunch and Parlett [1], using a mixture of 1×1 and 2×2 pivots chosen during the numerical factorization of A, allowing stable symmetric Gaussian elimination for indefinite systems. The method has been implemented by Duff and Reid [6] as a set of Fortran subroutines, currently available through the Harwell library under the name MA27, and is the one we used for our implementation.

During the assembly steps of the frontal method, one needs to access the entries of each element Hessian $H_i^{(k)}$. Since the element Hessians are stored in internal variables, each time we need one of them we must restore it in elemental variables. The user is then asked to write a code to perform operation (12), in addition to the operations (14). Numerical experience shows that this can often be done efficiently since (12) requires few floating point operations. Furthermore, we maintain the advantages of a reduction of storage for the Hessian $H^{(k)}$ and of an efficient updating technique.

One must point out however, that the current version of MA27 does not assume a finite-element representation for the matrix A but requires instead its storage in compact form. Consequently, for our experimental code, one had to assemble the Hessian, even though this type of method does not require it. This does not affect the viability of the frontal approach, or alter our conclusions. Moreover there are plans to introduce a new version of MA27 that will allow an elemental representation of the coefficient matrix (Iain Duff, private communication).

Given the Bunch-Parlett factorization of the Hessian (restricted to the free variables)

$$Z_C^{(k)T} H^{(k)} Z_C^{(k)} = L^{(k)} D^{(k)} L^{(k)T}, \tag{32}$$

where $Z_C^{(k)T} = Z(x_C^{(k)})^T$ is the orthogonal projector onto $C(x_C^{(k)})$, we check the 1x1 and 2x2 pivots stored in $D^{(k)}$ to decide whether it is positive definite, indefinite, or singular. Different strategies will be applied in these different cases.

If the restricted Hessian is positive definite, we can solve the Newton equations

$$Z_C^{(k)T} H^{(k)} Z_C^{(k)} z = -Z_C^{(k)T}\left[g^{(k)} + H^{(k)}(x_C^{(k)} - x^{(k)})\right] = -Z_C^{(k)T} \nabla m^{(k)}(x_C^{(k)}) \tag{33}$$

for the direction $z \in C(x_C^{(k)})$, using the MA27 solver.

If $Z_C^{(k)T} H^{(k)} Z_C^{(k)}$ is indefinite, the solution of (33) is no longer that of the sub-problem (18)–(19) restricted to $C(x_C^{(k)})$. Fortunately, the decomposition (32) allows us to compute a direction of negative curvature for this sub-problem.

To compute such a direction, we first chose λ, the most negative eigenvalue of $D^{(k)}$ and computed the corresponding eigenvector $v \in C(x_C^{(k)})$. The direction z would then be given at the cost of the backward substitution

$$z = L^{(k)^{-T}} v \tag{34}$$

and the corresponding curvature would be

$$z^T Z_C^{(k)T} H^{(k)} Z_C^{(k)} z = \lambda \|v\|^2. \tag{35}$$

Numerical experiments indicate a drawback of this method. When the restricted Hessian remains indefinite over a number of successive iterations, the directions z often lie in the same subspace and the number of iterations required to reach optimality is unacceptably high. To avoid this defect, when successive directions of negative curvature are encountered, instead of choosing the most negative eigenvalue of $D^{(k)}$, we cycle through its negative eigenvalues. By cycling we mean that we choose the negative eigenvalue ordered next to the one used at the previous iteration until we reach the last, in which case we repeat the cycle starting with the most negative again.

The last case to consider is when the restricted Hessian is singular and positive semidefinite, although we noticed that it occurs very rarely in practice. The strategy used is the one described by Conn and Gould [2], which consists of solving the linear system (33) if it is consistent, or finding a descent direction z otherwise, satisfying

$$Z_C^{(k)T} H^{(k)} Z_C^{(k)} z = 0 \text{ and } z^T Z_C^{(k)T} \nabla m^{(k)}(x_C^{(k)}) < 0. \tag{36}$$

Once z is known, the step $s^{(k)}$ is finally computed as

$$s^{(k)} = x_C^{(k)} - x^{(k)} + \min[1, \alpha^{(k)}][Z_C^{(k)T}]^+ z, \tag{37}$$

where $[Z_C^{(k)T}]^+$ is the Moore-Penrose generalized inverse of $Z_C^{(k)T}$ (that is the operator that completes a vector in $C(x_C^{(k)})$ with zeros to obtain a vector in $\mathbf{R}^n$), and where $\alpha^{(k)}$ is the largest admissible steplength for which the bounds (19) are satisfied at the point $x^{(k)} + s^{(k)}$.

6 Numerical experiments.

The test problems we used for our experiments come mainly from the set of functions used by Toint for testing partially separable codes. They are fully described in [15] and the numbering we use here refers to that paper. We considered the problems 10 (Rosenbrock function), 11 (Linear minimum surface), 16 (Boundary value problem), 17 (Broyden tridiagonal), 22 (Diagonal quadratic), 31 (Extended ENGVL1), 33 (Extended Freudenstein) and 36 (Cube problem). For these problems, we used the starting points as defined in [15]. We also considered five additional problems, described in the appendix to this paper, so as to allow for other sparsity structures in our test set. These problems are chosen as representative of a larger set used by the authors.

All the experiments were performed on the CRAY 2 supercomputer of Harwell Laboratory. Our code, called hereafter SBMIN, is written in Fortran 77 and compiled using the CFT77 Fortran compiler. All timings reported are CPU seconds.

The initial trust region radius was $\Delta^{(0)} = 0.1\|g^{(0)}\|_2$, and we chose $\mu = 0.25$, $\eta = 0.75$, $\gamma_0 = 1/\sqrt{10}$ and $\gamma_2 = \sqrt{10}$. The stopping criteria we used was based on the order of magnitude of the gradient (projected on the feasible domain); we required its norm to be smaller than 10^{-6}.

In the tables, the symbol * indicates that the trust region radius has become too small and that the routine has consequently decided to stop. One must point out, however, that in those cases, the projected gradient norm was of order 10^{-4}, or even 10^{-5}, and the failure should be attributed to numerical rounding errors preventing the accurate calculation of the ratio ρ (equation (20)).

For each test problem, we consider three different methods, conjugate gradients (cg), diagonally preconditioned conjugate gradients (pcg) and multifrontal (multif), to obtain an approximate solution to the sub-problem (18)–(19). We also consider three ways of computing the element Hessians: exact derivatives (exact), the Broyden-Fletcher-Goldfarb-Shanno update (BFGS) and the symmetric rank-one update (rk1). Specifically, in the latter two cases the i-th element Hessian, $H_i^{(k)}$, stored in terms of its internal variables, is updated from the BFGS formula

$$\hat{H}_i^{(k+1)} = \hat{H}_i^{(k)} + \frac{\hat{y}_i^{(k)} \hat{y}_i^{(k)T}}{\hat{y}_i^{(k)T} \hat{s}_i^{(k)}} - \frac{\hat{H}_i^{(k)} \hat{s}_i^{(k)} \hat{s}_i^{(k)T} \hat{H}_i^{(k)}}{\hat{s}_i^{(k)T} \hat{H}_i^{(k)} \hat{s}_i^{(k)}}, \tag{38}$$

or from the rank-one formula

$$\hat{H}_i^{(k+1)} = \hat{H}_i^{(k)} + \frac{\hat{r}_i^{(k)} \hat{r}_i^{(k)T}}{\hat{r}_i^{(k)T} \hat{s}_i^{(k)}}. \tag{39}$$

Here $\hat{s}_i^{(k)}$ and $\hat{y}_i^{(k)}$ are, respectively, the change in the internal variables $U_i(x^{(k+1)} - x^{(k)})$, and the change in the elemental gradient $\hat{g}_i^{(k+1)} - \hat{g}_i^{(k)}$, and $\hat{r}_i^{(k)}$ is defined as $\hat{y}_i^{(k)} - \hat{H}_i^{(k)} \hat{s}_i^{(k)}$. The

BFGS update is only performed for a given element if the new approximation can be ensured to be positive definite, and this is implemented by only allowing an update if the condition

$$\|\hat{y}_i^{(k)}\|^2 \leq 10^8 \hat{y}_i^{(k)T} \hat{s}_i^{(k)}$$

is satisfied. The rank-one update is only made when the correction has norm smaller than 10^8, i.e. when

$$\|\hat{r}_i^{(k)}\|^2 \leq 10^8 |\hat{r}_i^{(k)T} \hat{s}_i^{(k)}|.$$

The initial estimate of each element Hessian $\hat{H}_i^{(0)}$ is set to the identity matrix when updating schemes are used. This choice is considered satisfactory as the test problems are reasonably well scaled.

The figures we report in each Table are the number of function evaluations (f calls), the number of gradient evaluations (g calls), and the overall CPU-time (total cpu). Under the label "linear system stats" we give the number of conjugate gradient iterations in the case of the iterative method or information of the type *pd nc sc (ratio)* for the direct methods. Here *pd* is the number of positive definite linear systems solved, *nc* is the number of directions of negative curvature taken, *sc* is the number of singular (but consistent) linear systems solved and *ratio* gives an idea of the fill-in during the Gaussian elimination and is equal to the storage space needed to store the factors of the Gaussian decomposition divided by the space needed for the original matrix.

Table 1 presents the performance of the different methods on a set of 13 problems. Each problem was specified with 100 variables.

pb	Hessian	method	f calls	g calls	linear system stats	total cpu
10	exact	cg	441	289	2064	5.74
		pcg	492	259	2133	6.27
		multif	527	272	512 11 0 (1.00)	12.11
	BFGS	cg	617	380	2883	9.44
		pcg	603	368	2016	7.65
		multif	348	279	342 0 0 (1.00)	8.70
	rk1	cg	1047	617	3037	12.18
		pcg	1089	642	1755	11.27
		multif	1747	1021	562 1054 102 (1.00)	44.62
11	exact	cg	35	29	80	0.62
		pcg	38	31	71	0.85
		multif	15	14	15 0 0 (1.89)	0.44
	BFGS	cg	17	18	154	0.65
		pcg	21	19	135	0.80
		multif	17	18	17 0 0 (1.89)	0.56
	rk1	cg	65	35	178	1.35
		pcg	53	36	85	1.32
		multif	96	61	18 78 0 (1.89)	2.96

pb	Hessian	method	f calls	g calls	linear system stats	total cpu
16	exact	cg	3	4	254	0.94
		pcg	4	5	377	1.45
		multif	3	4	3 0 0 (1.00)	0.11
	BFGS	cg	20	20	2604	9.80
		pcg	17	17	2189	8.40
		multif	21	21	20 0 0 (1.00)	0.78
	rk1	cg	14	11	836	3.21
		pcg	8	7	329	1.38
		multif	6	6	4 1 0 (1.00)	0.20
17	exact	cg	7	8	29	0.17
		pcg	6	7	28	0.21
		multif	5	6	5 0 0 (1.00)	0.17
	BFGS	cg	11	9	30	0.24
		pcg	11	9	32	0.32
		multif	11	9	9 0 0 (1.00)	0.35
	rk1	cg	15	9	40	0.32
		pcg	11	9	35	0.33
		multif	26	9	8 16 0 (1.00)	0.83
22	exact	cg	6	7	8	0.05
		pcg	2	3	1	0.02
		multif	3	4	1 0 0 (1.00)	0.04
	BFGS	cg	22	15	76	0.37
		pcg	33	18	77	0.49
		multif	25	16	19 0 0 (1.00)	0.69
	rk1	cg	10	8	10	0.11
		pcg	10	8	4	0.10
		multif	6	4	1 0 0 (1.00)	0.08
31	exact	cg	8	9	21	0.07
		pcg	8	9	13	0.06
		multif	7	8	5 0 0 (1.00)	0.13
	BFGS	cg	13	11	31	0.15
		pcg	13	11	25	0.15
		multif	11	9	7 0 0 (1.00)	0.22
	rk1	cg	13	10	27	0.14
		pcg	25	10	25	0.20
		multif	15	10	6 5 0 (1.00)	0.31

pb	Hessian	method	f calls	g calls	linear system stats	total cpu
33	exact	cg	11	12	25	0.10
		pcg	7	8	18	0.07
		multif	10	11	4 0 0 (1.00)	0.13
	BFGS	cg	19	15	26	0.19
		pcg	17	13	23	0.18
		multif	17	13	6 0 0 (1.00)	0.24
	rk1	cg	17	13	28	0.17
		pcg	18	13	27	0.18
		multif	37	14	7 20 0 (1.00)	0.73
36	exact	cg	1029	618	7827	17.81
		pcg	1023	594	4030	11.74
		multif	944	562	932 7 0 (1.00)	22.42
	BFGS	cg	1515	963	9143	25.48
		pcg	1410	928	4449	17.00
		multif	1189	812	1176 1 0 (1.00)	29.75
	rk1	cg	3152	1824	9037	35.79
		pcg	3087	1804	4616	28.52
		multif	>10000	–	–	–
55	exact	cg	5	6	4	0.03
		pcg	5	6	4	0.04
		multif	5	6	2 0 0 (1.00)	0.08
	BFGS	cg	12	9	2	0.09
		pcg	13	10	9	0.12
		multif	13	10	3 0 0 (1.00)	0.17
	rk1	cg	22	10	14	0.15
		pcg	15	10	11	0.13
		multif	22	10	12 0 0 (1.00)	0.20
56	exact	cg	12	13	14	0.16
		pcg	12	13	0	0.22
		multif	12	13	11 0 0 (1.00)	0.39
	BFGS	cg	19	20	430	1.80
		pcg	19	20	452	2.06
		multif	19	20	13 0 0 (1.00)	0.59
	rk1	cg	39	25	17	0.69
		pcg	32	22	26	0.70
		multif	33	22	1 22 0 (1.01)	0.98

pb	Hessian	method	f calls	g calls	linear system stats	total cpu
57	exact	cg	107	68	1380	5.83
		pcg	94	57	741	3.94
		multif	15	16	11 0 0 (1.00)	0.47
	BFGS	cg	75	53	942	4.56
		pcg	332	230	741	9.34
		multif	24	22	16 0 0 (1.00)	0.82
	rk1	cg	81	58	1382	6.28
		pcg	238	168	605	6.99
		multif	24	22	16 0 0 (1.00)	0.83
59	exact	cg	8	7	16	0.12
		pcg	10	9	18	0.16
		multif	12	9	4 5 0 (2.68)	0.45
	BFGS	cg	13	12	21	0.29
		pcg	13	12	14	0.28
		multif	13	12	12 0 0 (2.68)	0.70
	rk1	cg	18	8	31	0.33
		pcg	13	10	7	0.24
		multif	21	10	6 11 0 (2.75)	0.94
61	exact	cg	13	14	56	0.35
		pcg	11	12	25	0.21
		multif	11	12	10 0 0 (1.00)	0.54
	BFGS	cg	33	25	136	1.05
		pcg	26	20	66	0.70
		multif	26	20	18 0 0 (1.00)	1.19
	rk1	cg	45	22	131	1.23
		pcg	56	25	45	1.08
		multif	116	69	20 88 0 (1.00)	6.18

Table 1 : Performance of the methods on test problems with 100 variables.

This table indicates the viability of the multifrontal method in the context of partially separable optimization and more than that, we already see that the method seems to be really competitive with the iterative schemes in some cases. It is not rare that the number of function and gradient evaluations is significantly reduced and this can lead to significant improvements in terms of computation time.

Two special points in these results are worth more comment.

1. We first observe that, on problem 22 with the multifrontal solver, BFGS requires the solution of 19 linear systems while rk1 only requires 1! This behaviour is explained by the conjunction of the quadratic termination properties of the rk1 update and the particular structure of problem 22. This structure is such that all element Hessians are constant diagonal 3×3 matrices. Because rk1 needs at most p steps to obtain an exact $p \times p$ con-

stant Hessian, the exact Hessian is obtained after 3 steps in problem 22. These first 3 steps did not require the complete solution of (33) because the test (25) was satisfied. A fourth iteration and a single linear system solution are then all that is needed to minimize the quadratic objective exactly. Since the BFGS update does not enjoy similar quadratic termination properties, more iterations are required to form a good approximation of the Hessian and to converge.

2. We also note that, for problem 36 using BFGS, the multifrontal method finds one indefinite system. This is very surprising, as the BFGS update ensures positive definiteness of the Hessian approximations. This last property, although true in exact arithmetic, can however be violated due to rounding errors in the case where the matrix to update has a condition number of the order of the inverse of machine precision. This is what happens in problem 36: the conditioning of some element Hessian matrices gradually builds up and finally exceeds 10^{17}!

   When this situation occurs, it seems inadvisable to keep on updating an indefinite matrix with the BFGS update. We therefore decided to reinitialize the element Hessian approximations to the identity matrix, in effect restarting the algorithm. We are well aware that this technique is merely a "quick fix", and that a more sophisticate procedure is desirable. Finding such a procedure might however be difficult, because one may wish to detect the bad conditioning of the element Hessians before negative curvature is actually produced, while maintaining an unfactored element Hessian representation, as needed in the partitioned updating framework.

In order to investigate more carefully the relative performance of the methods, we increased the dimension n up to a maximum of 5000. The results of those experiments are given in Tables 2 to 7 for a subset of our test problems. The problems were chosen as being fairly representative of the larger set.

pb	Hessian	method	f calls	g calls	linear system stats	total cpu
961	exact	cg	507	453	860	84.45
		pcg	179	130	327	42.39
		multif	26	20	26 0 0 (4.17)	12.53
	BFGS	cg	52	38	565	25.30
		pcg	117	80	550	41.01
		multif	26	21	26 0 0 (4.17)	13.37
	rk1	cg	613	379	771	156.38
		pcg	450	280	349	104.35
		multif	–	–	–	>1200
4900	exact	cg	–	–	–	>1200
		pcg	574	435	848	713.70
		multif	36	29	36 0 0 (6.43)	108.26
	BFGS	cg	133	97	1422	352.69
		pcg	–	–	–	>1200
		multif	32	25	32 0 0 (6.43)	104.66
	rk1	cg	–	–	–	>1200
		pcg	–	–	–	>1200
		multif	–	–	–	>1200

Table 2 : Performance of the methods on the test problem 11 for increasing dimensions.

pb	Hessian	method	f calls	g calls	linear system stats	total cpu
1000	exact	cg	2	3	256	9.18
		pcg	2	3	612	22.31
		multif	5	6	5 0 0 (1.00)	1.75
	BFGS	cg	15	15	10770	1938.77
		pcg	–	–	–	>1200
		multif	37	32	35 1 0 (1.00)	14.53
	rk1	cg	12	8	823	31.97
		pcg	7	6	3085	117.50
		multif	11	11	3 7 0 (1.00)	4.07
5000	exact	cg	2	3	1151	204.75
		pcg	2	3	1830	332.50
		multif	7	8	7 0 0 (1.00)	12.19
	BFGS	cg	–	–	–	>1200
		pcg	–	–	–	>1200
		multif	–	–	–	>1200
	rk1	cg	14	9	184	43.07
		pcg	18	10	124	39.07
		multif	58	34	11 46 1 (1.01)	105.62

Table 3 : Performance of the methods on the test problem 16 for increasing dimensions.

pb	Hessian	method	f calls	g calls	linear system stats	total cpu
1000	exact	cg	13	14	0	1.23
		pcg	13	14	0	2.33
		multif	13	14	0 0 0 (–)	1.23
	BFGS	cg	21	22	1453	54.41
		pcg	21	22	2628	99.46
		multif	21	22	12 0 0 (1.00)	5.97
	rk1	cg	36	24	17	6.38
		pcg	35	26	30	7.81
		multif	42	27	0 30 0 (1.00)	21.10
5000	exact	cg	14	15	0	6.62
		pcg	14	15	0	12.49
		multif	14	15	0 0 0 (–)	6.56
	BFGS	cg	22	23	2170	396.23
		pcg	–	–	–	>1200
		multif	22	23	11 0 0 (1.00)	28.81
	rk1	cg	36	25	13	29.79
		pcg	38	28	35	42.60
		multif	–	–	–	>1200

Table 4 : Performance of the methods on the test problem 56 for increasing dimensions.

pb	Hessian	method	f calls	g calls	linear system stats	total cpu
1000	exact	cg	143	93	1964	81.67
		pcg	206	128	955	64.91
		multif	17	18	10 0 0 (1.00)	9.45
	BFGS	cg	210	147	2177	102.33
		pcg	560	410	860	134.44
		multif	19	17	15 0 0 (1.00)	14.28
	rk1	cg	218	152	2103	100.51
		pcg	673	476	911	152.97
		multif	19	17	15 0 0 (1.00)	14.18
5000	exact	cg	146	94	1774	382.91
		pcg	154	99	735	248.88
		multif	18	19	10 0 0 (1.00)	158.42
	BFGS	cg	289	205	3235	751.54
		pcg	–	–	–	>1200
		multif	20	18	16 0 0 (1.00)	259.90
	rk1	cg	222	158	2329	559.15
		pcg	–	–	–	>1200
		multif	20	18	16 0 0 (1.00)	252.50

Table 5 : Performance of the methods on the test problem 57 for increasing dimensions.

pb	Hessian	method	f calls	g calls	linear system stats	total cpu
1000	exact	cg	10	7	23	1.59
		pcg	14	11	31	2.31
		multif	40	24	6 31 0 (12.13)	52.28
	BFGS	cg	14	13	44	3.51
		pcg	12	11	10	2.33
		multif	13	12	11 0 0 (12.13)	15.97
	rk1	cg	24	17	12	5.02
		pcg	16	13	6	2.82
		multif	36	17	2 24 0 (12.04)	36.00
5000	exact	cg	19	12	44	15.62
		pcg	10	8	16	7.10
		multif	16	10	4 10 0 (52.39)	488.78
	BFGS	cg	15	14	49	19.93
		pcg	13	12	15	14.05
		multif	15	14	14 0 0 (51.98)	447.67
	rk1	cg	38	19	17	51.98
		pcg	19	13	6	16.35
		multif	40	21	2 29 0 (52.34)	1070.84

Table 6 : Performance of the methods on the test problem 59 for increasing dimensions.

pb	Hessian	method	f calls	g calls	linear system stats	total cpu
1000	exact	cg	14	15	64	4.10
		pcg	11	12	24	2.32
		multif	12	13	10 0 0 (1.00)	10.81
	BFGS	cg	36	25	151	12.19
		pcg	29	22	61	* 7.80
		multif	28	22	18 0 0 (1.00)	22.37
	rk1	cg	57	27	175	15.50
		pcg	46	25	44	* 9.42
		multif	–	–	–	>1200
5000	exact	cg	14	15	56	* 23.58
		pcg	11	12	23	16.03
		multif	12	13	10 0 0 (1.00)	171.92
	BFGS	cg	41	29	119	* 62.12
		pcg	28	21	67	42.90
		multif	–	–	–	>1200
	rk1	cg	40	24	93	53.55
		pcg	75	34	100	82.58
		multif	–	–	–	>1200

Table 7 : Performance of the methods on the test problem 61 for increasing dimensions.

When exact derivatives are available, the multifrontal method seems competitive with respect to the conjugate gradient type algorithms in many cases. It appears to be also the case when the Broyden-Fletcher-Goldfarb-Shanno update is applied. If the symmetric rank-one update is used however, conjugate gradient methods are preferable.

We observe the excellent performances of the multifrontal method when the quadratic model is convex (see Table 5 for instance). However, if the fill-in of the factorization is too high, the number of operations for the Gaussian elimination is dominant and the multifrontal scheme is no longer competitive. A good example of this is shown in Table 6 where the sparsity pattern of the Hessian is randomly generated.

We observe particularly poor performances of the direct method when directions of negative curvature are taken. This happens, obviously, more often with the symmetric rank-one update of the Hessian. Illustration of this behaviour can be found in Tables 3 and 6. The proposed strategy seems to be clearly inefficient and other strategies must definitely be found to handle such cases, if direct methods are to be used.

We note that the negative curvature directions generated by the rank-one update are always taken into account when using our multifrontal approach, in contrast with the (preconditioned) conjugate gradient technique. This last calculation only considers the first few Krylov subspaces spanned by the gradient and its successive images by the Hessian approximation, especially in the early iterations when this approximation may be quite poor and the trust region radius small. The comparison of the performance of the rank-one update with the (preconditioned) conjugate gradient and multifrontal schemes, in cases where negative curvature is encountered, tends to show that this possible neglect of negative curvature in the early stages of the computation might be advantageous.

We also wanted our algorithm to converge to a local minimum from any starting point. We therefore ran several additional tests using the same objective functions as before, but with different starting points. These experiments did not affect the conclusions above.

Finally, in Conn, Gould and Toint [4], the simplest of the updating schemes, the symmetric rank-one method, appeared to perform better than the BFGS method for many of the problems tested. In the context of partial separability, this conclusion does not appear to apply. One could attempt to explain this phenomenon by the fact that it is not uncommon for the projection of successive search directions into the range space of certain elements to be close to linear dependence, despite the independence of the overall search directions. Linear dependence of the search directions can be highly undesirable for the rank-one update.

7 Conclusions.

The numerical experiments show that the use of a direct method instead of an iterative one can sometimes lead to very significant improvements in terms of computation time. They also clearly demonstrate that the improvements can be achieved only in cases where the quadratic model of the function at the current iterate is convex (or at least not too often non-convex) and the structure of the Hessian is sufficiently regular to avoid high fill-in during the factorization.

When the approximation of the function Hessian is indefinite, the use of directions of negative curvature inhibits fast convergence of direct methods and can even lead to a dramatic increase in computation time. This conclusion convinced the authors that in this particular case, other strategies must definitely be used.

The main conclusion is that an efficient code for partially separable optimization must provide a choice of methods more adapted to the specific problem being solved. The authors intend to provide such a code through the Harwell library in the future.

Acknowledgements.

The project has been financially supported by the Belgian National Fund for Scientific Research, Harwell Laboratory (U.K.A.E.A.) and by grant A8639 from the Natural Sciences and Engineering Research Council of Canada. This support is gratefully acknowledged.

Appendix

The five problems that we added to introduce other types of sparsity structure are now given. For each of them we mention (a) the element functions, (b) any bounds on the variables and (c) the starting point.

Test problem 55

(a) $f_i(x) = (x_i^2 + x_n^2)^2 - 4x_i + 3, \quad (i = 1, ..., n-1).$

(c) $x = (1, 1, \ldots, 1).$

Test problem 56

(a) $f_i(x) = (x_i + x_{i+1})e^{-x_{i+2}(x_i + x_{i+1})}, \quad (i = 1, \ldots, n-2).$

(b) $x_i \geq 0, \quad (i = 1, \ldots, n)$

(c) $x = (1, 1, \ldots, 1)$

Test problem 57

(a) $f_i(x) = (x_i + x_{i+1} + x_n)^4, \quad (i = 1, \ldots, n-2)$
$f_{n-1}(x) = (x_1 - x_2)^2,$
$f_n(x) = (x_{n-1} - x_n)^2.$

(c) $x = (1, -1, 1, -1, \ldots).$

Test problem 59

(a) $f_i(x) = x_i^2 e^{-x_{j_i}} \quad (i = 1, \ldots, n)$,
$f_i(x) = x_{i-n}^2 e^{-x_{j_i}} \quad (i = n+1, \ldots, 2n)$,
where the indices j_i are randomly generated between 1 and n in order by subroutine FA04BS of the Harwell Subroutine Library, starting with the default seed, but with the provision that any j_i equal to i is rejected and the next random number in the sequence taken.

(c) $x = (1, -1, 1, -1, \ldots)$.

Test problem 61

(a) $f_i(x) = (x_i^2 + 2x_{i+1}^2 + 3x_{i+2}^2 + 4x_{i+3}^2 + 5x_n^2)^2 - 4x_i + 3, \quad (i = 1, \ldots, n-4)$.

(c) $x = (1, 1, \ldots, 1)$.

References

[1] Bunch, J.R. and Parlett, B.N. (1971). Direct methods for solving symmetric indefinite systems of linear equations. SIAM J. Numer. Anal. 8, 639-655.

[2] Conn, A.R. and Gould, N.I.M. (1984). On the location of directions of infinite descent for nonlinear programming algorithms. SIAM J. Numer. Anal., Vol. 21, No. 6, 302-325.

[3] Conn, A.R., Gould, N.I.M. and Toint, Ph.L. (1988). Global convergence of a class of trust region algorithms for optimization with simple bounds. SIAM J. Numer. Anal., Vol. 25, No. 2, 433-460.

[4] Conn, A.R., Gould, N.I.M. and Toint, Ph.L. (1988). Testing a class of methods for solving minimization problems with simple bounds on the variables. Mathematics of Computation, vol. 50(182), pp. 399–430.

[5] Dennis, J.E. and Schnabel, R.B. (1983). Numerical methods for unconstrained optimization and nonlinear equations. Prentice-Hall, New Jersey.

[6] Duff, I.S. and Reid, J.K. (1982). MA27 - A set of Fortran subroutines for solving sparse symmetric sets of linear equations. Report HL82/2225 (C13), Harwell Laboratory, Oxfordshire, England.

[7] Duff, I.S. and Reid, J.K. (1983). The multifrontal solution of indefinite sparse symmetric linear equations. ACM Transactions on Mathematical Software, Vol. 9, No. 3, 302-325.

[8] Griewank, A. and Toint, Ph.L. (1982). Partitioned variable metric updates for large structured optimization problems, Numerische Mathematik, vol. 39, 119-137.

[9] Griewank, A. and Toint, Ph.L. (1984). Numerical experiments with partially separable optimization problems, in Numerical Analysis: Proceedings Dundee 1983 (D.F. Griffiths,ed.), Lecture Notes in Mathematics 1066, Springer Verlag, Berlin, 203-220.

[10] Irons, B.M. (1970). A frontal solution program for finite-element analysis. Int. J. Numer. Meth. Eng. 2, 5-32.

[11] Moré, J.J. (1977). The Levenberg-Marquardt algorithm : implementation and theory. Numerical Analysis, G.A. Watson, ed., Lecture Notes in Math. 630, Springer-Verlag, Berlin, 105-116.

[12] Steihaug, T. (1983). The conjugate gradient method and trust regions in large scale optimization. SIAM Journal on Numerical Analysis, Vol.20, No. 3, 626-637.

[13] Stoer, J. (1983). Solution of large linear systems of equations by conjugate gradient type methods, in Mathematical Programming : The State of the Art (Bonn 1982), A. Bachem, M. Grötschel and B. Korte (eds.), Springer Verlag, Berlin, 540-565.

[14] Toint, Ph.L. (1981). Towards an efficient sparsity exploiting Newton method for minimization, in Sparse Matrices and their Uses (I. S. Duff, ed.), Academic Press, London.

[15] Toint, Ph.L. (1983). Test problems for partially separable optimization and results for the routine PSPMIN. Report 83/4 of the FUNDP, Namur.