On Iterated-Subspace Minimization Methods for Nonlinear Optimization

by A. R. Conn[1], Nick Gould[2], A. Sartenaer[3] and Ph. L. Toint[3]

January 16, 2018

Abstract. We consider a class of Iterated-Subspace Minimization (ISM) methods for solving large-scale unconstrained minimization problems. At each major iteration of such a method, a low-dimensional manifold, the iterated subspace, is constructed and an approximate minimizer of the objective function in this manifold then determined. The iterated subspace is chosen to contain vectors which ensure global convergence of the overall scheme and may also contain vectors which encourage fast asymptotic convergence. We demonstrate that this approach can sometimes be very advantageous and indicate the general performance on a collection of large problems. Moreover, comparisons with a limited memory approach and LANCELOT are made.

[1] IBM T.J. Watson Research Center, P.O.Box 218, Yorktown Heights, NY 10598, USA
Email : arconn@watson.ibm.com

[2] Central Computing Department, Rutherford Appleton Laboratory, Chilton, Oxfordshire, OX11 0QX, England
Email : n.gould@letterbox.rl.ac.uk
Current reports available by anonymous ftp from the directory "pub/reports" on joyous-gard.cc.rl.ac.uk (internet 130.246.9.91)

[3] Department of Mathematics, Facultés Universitaires ND de la Paix, 61 rue de Bruxelles, B-5000 Namur, Belgium, EC
Email : as@math.fundp.ac.be or pht@math.fundp.ac.be
Current reports available by anonymous ftp from the directory "pub/reports" on thales.math.fundp.ac.be (internet 138.48.4.14)

Keywords: Unconstrained optimization, large-scale computation.

Mathematics Subject Classifications: 65K05, 90C30

Running title: Iterated-Subspace Minimization

This work was supported by the Belgian National Fund for Scientific Research. The research of Conn and Toint was supported in part by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Air Force Office of Scientific Research under Contract No F49620-91-C-0079. The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

On Iterated-Subspace Minimization Methods for Nonlinear Optimization

A. R. Conn, Nick Gould, A. Sartenaer and Ph. L. Toint

January 16, 2018

Abstract

We consider a class of Iterated-Subspace Minimization (ISM) methods for solving large-scale unconstrained minimization problems. At each major iteration of such a method, a low-dimensional manifold, the iterated subspace, is constructed and an approximate minimizer of the objective function in this manifold then determined. The iterated subspace is chosen to contain vectors which ensure global convergence of the overall scheme and may also contain vectors which encourage fast asymptotic convergence. We demonstrate that this approach can sometimes be very advantageous and indicate the general performance on a collection of large problems. Moreover, comparisons with a limited memory approach and LANCELOT are made.

1 Introduction

In this paper, we consider finding a local solution of the unconstrained minimization problem,

$$\underset{\boldsymbol{x}\in\Re^n}{\text{minimize}} \quad f(\boldsymbol{x}), \tag{1.1}$$

where we assume, for simplicity, that the objective function $f \in C^2$. We are particularly interested in the case where n is sufficiently large that methods appropriate for small problems — such as those which might maintain a dense factorization of a suitable approximation of the Hessian matrix, see, for example, Gill *et al.* (1981), Dennis and Schnabel (1983) and Fletcher (1987) — are impractical. We are especially interested in solving large problems for which exact second derivatives are explicitly available. At variance with a popular misconception, such problems frequently arise in practice, as is evident when one considers, amongst others, the large CUTE collection (see Bongartz *et al.*, 1995).

We are primarily concerned with the commonly occurring case in which the cost of evaluating the value of the objective function and its derivatives, at a given point $\boldsymbol{x}$, is less significant than the cost of solving, for instance, the Newton equations. Our experience with the large-scale nonlinear optimization package LANCELOT (see Conn *et al.*, 1992) has been that it is the linear-algebra cost which tends to dominate when solving a significant number of widely differing application problems (see, Conn *et al.*, 1993 and Conn *et al.*, 1996). Thus, it would appear desirable in these cases to attempt to reduce the linear-algebra costs, even if this results in an increase in the number of objective function evaluations.

The most common methods for unconstrained minimization either determine a search direction followed by a linesearch or use the trust-region approach (see, for example, Dennis and Schnabel, 1983). In the former case, a simple model of the underlying objective function is constructed in order to determine the search direction. By contrast, in the latter case, an approximate minimizer of the model within a restricted domain (the trust region) is determined. This model minimizer is then used as a prediction of the actual minimizer of the true objective. In a trust-region method, success of this process is measured by comparing the model and true function values at the predicted minimizer. In linesearch methods, the true function is used to establish a step size. Thus, both of these approaches may be considered to perform their multi-dimensional work with respect to a model whilst probing the true function uni-dimensionally. Of course, the model does make use of the true function and perhaps its derivatives — maybe at more than a single point.

In this paper, we take the view that the above schemes are quite wasteful, given the amount of information that may have been accrued during the (approximate) minimization of the model. In particular, the model may have been sampled in a number of potentially interesting directions, of which only the aggregate direction is normally considered to be of significance.

We also believe that, provided function and derivative values are inexpensive to compute relative to the linear-algebra costs, an (approximate) low-dimensional minimization is a relatively simple calculation. Indeed, we feel that there is high-quality, robust, general-purpose software readily available for the small-scale unconstrained minimization problem, and that such software is normally capable of solving problems of modest dimensions - say up to a hundred variable problems - extremely fast on current workstations *provided that* function evaluation is cheap. Of course there are, and will continue to be, small-scale problems which are challenging, because they are so nonlinear that algorithms implemented in fixed, finite precision arithmetic are unsuccessful, but in our experience such examples occur rarely in practice.

Thus, in this paper, we propose methods which aim to investigate the *true* objective function in a space larger than the one-dimensional space which is normally associated with linesearch or trust-region methods. We do this knowing that, so long as the space is relatively modest, the approximate multi-dimensional minimization will still be a manageable calculation. Moreover, by carefully choosing the space that we investigate, we hope to reduce significantly the linear-algebra costs while still maintaining global, and fast asymptotic, convergence.

Previous work along the lines we are considering includes Cragg and Levy (1969) who propose minimizing in a k-dimensional subspace, which includes the steepest descent direction. The remaining k-1 directions are the steps from the k-1 previous outer iterations. See also, for example, Dennis and Turner (1987), Dixon *et al.* (1984), Miele and Cantrell (1969), and Vinsome (1976). The first minimizes a convex quadratic on a subspace by adding, at each iteration, an extra vector that is not dependent on the existing subspace. They thus provide a uniform framework for a wide class of conjugate directions. The second minimizes the objective function over a grid of 4096 points, generated from a four-dimensional subspace. This subspace is defined using the steepest descent direction, the Newton direction, and two others directions which are combinations of the two previous steps.

A particular form of these ideas has been given by Saad (1990) for the solution of nonlinear systems of equations. Here, a sequence of iterates are generated as least-squares solutions to the equations in suitable Krylov subspaces. The principal difference is that, in

Saad's proposal, the entire Krylov subspace generated is used, while, as we shall see, this is in general quite unnecessary. Another example, that also is in the context of nonlinear equations is that of Kaporin and Axelsson (1995). Although developed independently, many of the ideas are quite similar to those of this paper, but the emphasis is rather different.

Given an initial estimate of the solution to (1.1), $\boldsymbol{x}^{(0)}$, and an iteration count, k, set initially to zero, a prototype algorithm for the proposed method might be as follows:

1. Stop with the solution estimate $\boldsymbol{x}^{(k)}$ if convergence tests are satisfied.
2. Determine a full-rank subspace matrix $\boldsymbol{S}^{(k)} \in \Re^{n \times s^{(k)}}$, where $s^{(k)} \ll n$.
3. Approximately solve the $s^{(k)}$-dimensional minimization problem

$$\underset{\boldsymbol{y} \in \Re^{s^{(k)}}}{\text{minimize}} \quad f(\boldsymbol{x}^{(k)} + \boldsymbol{S}^{(k)}\boldsymbol{y}), \tag{1.2}$$

set

$$\boldsymbol{x}^{(k+1)} = \text{ (approximate) } \underset{\boldsymbol{y} \in \Re^{s^{(k)}}}{\arg\min} \quad f(\boldsymbol{x}^{(k)} + \boldsymbol{S}^{(k)}\boldsymbol{y}), \tag{1.3}$$

replace k by $k+1$ and return to step 1.

We refer to such a method as *Iterated-Subspace Minimization* or ISM for short. This is, of course, a multi-dimensional subspace analog of the unidimensional-subspace linesearch method.

We are interested in the following issues:

- What is a good choice for $s^{(k)}$?
- How do we determine the *Iterated-Subspace* matrix $\boldsymbol{S}^{(k)}$?
- What do we mean by "approximate" in the problem (1.3)?
- Are there methods which are particularly appropriate for solving (1.2)?
- What can we say about the convergence of such a method?
- If we can establish convergence, what can we say about its asymptotic rate?

In this paper, we make preliminary attempts to answer all of these questions.

We will use the following notation. Bold lower and upper case Roman letters indicate vectors and matrices, respectively, while Greek and normal Roman letters denote scalars. Script style letters are index sets. A superscript (k) indicates a quantity which occurs at the k-th iteration or which is evaluated at $\boldsymbol{x}^{(k)}$.

We let $\boldsymbol{g}(\boldsymbol{x})$ and $\boldsymbol{H}(\boldsymbol{x})$, respectively, indicate the gradient, $\nabla_x f(\boldsymbol{x})$, and Hessian matrix, $\nabla_{xx} f(\boldsymbol{x})$, of the objective function. We define

$$f_s^{(k)}(\boldsymbol{y}) \stackrel{\text{def}}{=} f(\boldsymbol{x}^{(k)} + \boldsymbol{S}^{(k)}\boldsymbol{y}), \tag{1.4}$$

$\boldsymbol{g}_s^{(k)}(\boldsymbol{y}) \stackrel{\text{def}}{=} \nabla_y f_s^{(k)}(\boldsymbol{y})$ and $\boldsymbol{H}_s^{(k)}(\boldsymbol{y}) \stackrel{\text{def}}{=} \nabla_{yy} f_s^{(k)}(\boldsymbol{y})$, and will make use of the derivative identities

$$\boldsymbol{g}_s^{(k)}(\boldsymbol{y}) = \boldsymbol{S}^{(k)T}\boldsymbol{g}(\boldsymbol{x}^{(k)} + \boldsymbol{S}^{(k)}\boldsymbol{y}) \tag{1.5}$$

and

$$\boldsymbol{H}_s^{(k)}(\boldsymbol{y}) = \boldsymbol{S}^{(k)T}\boldsymbol{H}(\boldsymbol{x}^{(k)} + \boldsymbol{S}^{(k)}\boldsymbol{y})\boldsymbol{S}^{(k)}. \tag{1.6}$$

The paper is organised as follows. In Section 2, we consider the convergence of the algorithm given in the introduction. We discuss a number of ISM methods in Section 3, and we report on some preliminary numerical experience when solving some relatively large test examples, from the CUTE test suite (see Bongartz *et al.*, 1995), in Section 4. Possible extensions, to the cases where there are linear or nonlinear constraints present, are given in our concluding Section 5, where we also offer our perspectives of this and future work.

2 Convergence of a general algorithm

Global convergence of the above scheme can be guaranteed under fairly general assumptions. Suppose that we are able to pick consecutive iterates $\boldsymbol{x}^{(k)}$ and $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{S}^{(k)}\boldsymbol{y}^{(k)}$ for which the Goldstein (1964) conditions

$$f^{(k)} + \beta \boldsymbol{g}^{(k)T}\boldsymbol{S}^{(k)}\boldsymbol{y}^{(k)} \leq f^{(k+1)} \leq f^{(k)} + \alpha \boldsymbol{g}^{(k)T}\boldsymbol{S}^{(k)}\boldsymbol{y}^{(k)}, \tag{2.1}$$

for some $0 < \alpha \leq \beta < 1$, are satisfied, where $\boldsymbol{y}^{(k)}$ is the approximate solution of (1.2). Suppose, furthermore, that

$$\frac{-\boldsymbol{g}^{(k)T}\boldsymbol{S}^{(k)}\boldsymbol{y}^{(k)}}{\|\boldsymbol{S}^{(k)T}\boldsymbol{g}^{(k)}\|_2\|\boldsymbol{y}^{(k)}\|_2} \geq \epsilon, \tag{2.2}$$

for some $\epsilon > 0$. Then the ISM algorithm from Section 1 is globally convergent to a stationary point for the problem (1.1) from any starting point so long as f is bounded from below and has a Lipschitz-continuous gradient (see, for example, Dennis and Schnabel, 1983, Theorem 6.3.3).

Using the definitions (1.4) and (1.5), we may write (2.1) as

$$f_s^{(k)}(\boldsymbol{0}) + \beta \boldsymbol{g}_s^{(k)}(\boldsymbol{0})^T\boldsymbol{y}^{(k)} \leq f_s^{(k)}(\boldsymbol{y}^{(k)}) \leq f_s^{(k)}(\boldsymbol{0}) + \alpha \boldsymbol{g}_s^{(k)}(\boldsymbol{0})^T\boldsymbol{y}^{(k)} \tag{2.3}$$

and ensure that (2.2) is satisfied by requiring that

$$\frac{-\boldsymbol{g}_s^{(k)}(\boldsymbol{0})^T\boldsymbol{y}^{(k)}}{\|\boldsymbol{g}_s^{(k)}(\boldsymbol{0})\|_2\|\boldsymbol{y}^{(k)}\|_2} \geq \epsilon. \tag{2.4}$$

This is relevant as now the global convergence conditions may be verified in terms of the inner-minimization function f_s and its gradient. Similar global convergence results can be obtained if we replace condition (2.1) by the Armijo (1966) backtracking strategy (see Bertsekas, 1982, Section 1.3). It may, however, be difficult to design general algorithms which ensure that conditions (2.3) and (2.4) are satisfied on exit from the inner minimization. Thus, it may be preferable to impose extra conditions on the iterates generated during the inner minimization to ensure overall global convergence. With this in mind, suppose that we can find any point $\boldsymbol{y}_s^{(k)}$ for which

$$f_s^{(k)}(\boldsymbol{0}) + \beta \boldsymbol{g}_s^{(k)}(\boldsymbol{0})^T\boldsymbol{y}_s^{(k)} \leq f_s^{(k)}(\boldsymbol{y}_s^{(k)}) \leq f_s^{(k)}(\boldsymbol{0}) + \alpha \boldsymbol{g}_s^{(k)}(\boldsymbol{0})^T\boldsymbol{y}_s^{(k)} \tag{2.5}$$

and

$$\frac{-\boldsymbol{g}_s^{(k)}(\boldsymbol{0})^T\boldsymbol{y}_s^{(k)}}{\|\boldsymbol{g}_s^{(k)}(\boldsymbol{0})\|_2\|\boldsymbol{y}_s^{(k)}\|_2} \geq \epsilon \tag{2.6}$$

are satisfied. Suppose, furthermore, that we terminate the inner minimization at a point $\boldsymbol{y}^{(k)}$ for which

$$f_s^{(k)}(\mathbf{0}) - f_s^{(k)}(\boldsymbol{y}^{(k)}) \geq \tau(f_s^{(k)}(\mathbf{0}) - f_s^{(k)}(\boldsymbol{y}_s^{(k)})), \tag{2.7}$$

where $\tau > 0$. Then it is easy to show that this scheme is globally convergent under the same conditions as stated above. The advantage here is that the tests (2.5) and (2.6) need only be satisfied at an intermediate point to ensure convergence. Typically, the first inner-iterate provides such a point for carefully chosen subspaces and minimizers. For example, if the subspace contains the steepest-descent direction and the inner minimization starts by performing a linesearch in this direction, the resulting first inner-iterate satisfies (2.5) and (2.6). Similarly, if the subspace contains a (modified) truncated-Newton direction and the inner minimization starts by performing a linesearch in this direction, the same conclusion is true.

3 Computational Variants

We consider it important from a practical point of view to require that $\boldsymbol{S}^{(k)}$ contains at least two components,

- a gradient-related direction, such as $-\boldsymbol{g}^{(k)}$, to encourage global convergence, and
- a Newton-related direction, such as might be computed by a truncated-Newton method, to encourage fast asymptotic convergence, with safeguards to account for indefiniteness.

Of course, these components play a key role in dog-leg trust-region methods (see, for example, Powell, 1970). Additional components have the advantage of enlarging the subspace searched, but the disadvantage of increasing the overheads in solving the $s^{(k)}$-dimensional subspace minimization problem. In this section, we consider various possible ways of choosing the enlarged iterated subspace by choosing suitable conjugate directions.

3.1 Conjugate gradients

An appealing choice of $\boldsymbol{S}^{(k)}$ may be obtained by picking the columns of $\boldsymbol{S}^{(k)}$ as a set of $\boldsymbol{H}^{(k)}$-conjugate directions, especially if these directions are generated by a conjugate-gradient (CG) method. In practise one would typically precondition first.

Suppose that $\boldsymbol{H}^{(k)}$ is positive definite. Let $\phi^{(k)}(\boldsymbol{x}^{(k)} + \boldsymbol{p})$ be the quadratic model,

$$\phi^{(k)}(\boldsymbol{x}^{(k)} + \boldsymbol{p}) = f^{(k)} + \boldsymbol{p}^T\boldsymbol{g}^{(k)} + \tfrac{1}{2}\boldsymbol{p}^T\boldsymbol{H}^{(k)}\boldsymbol{p}, \tag{3.1}$$

of $f(\boldsymbol{x}^{(k)}+\boldsymbol{p})$ about $\boldsymbol{x}^{(k)}$. The preconditioned conjugate-gradient method (see, for example, Hestenes and Stiefel, 1952 and Golub and Loan, 1989, Section 10.3) is an iterative method which may be used to calculate the smallest value of (3.1). It is well known that the solution, $\boldsymbol{p}_n$, to this problem is the Newton direction.

A *preconditioner* $\boldsymbol{P}^{(k)}$ is usually an easily invertible approximation to $\boldsymbol{H}^{(k)}$. We shall insist that $\boldsymbol{P}^{(k)}$ has a uniformly bounded condition number. The j-th step of the preconditioned conjugate-gradient method determines the smallest value of (3.1) in the Krylov subspace spanned by the vectors $\{-((\boldsymbol{P}^{(k)})^{-1}\boldsymbol{H}^{(k)})^i(\boldsymbol{P}^{(k)})^{-1}\boldsymbol{g}^{(k)}\}_{i=0}^{j}$. Conjugacy properties ensure that each successive step may be accomplished by a univariate minimization of (3.1) in the direction $\boldsymbol{s}_j$; the vectors $\{\boldsymbol{s}_j\}$ are conjugate and are recurred from step to step.

Significantly from our point of view, the first such vector, $\boldsymbol{s}_0 = -(\boldsymbol{P}^{(k)})^{-1}\boldsymbol{g}^{(k)}$. In exact arithmetic the method would terminate with the Newton direction, $\boldsymbol{p}_n$, after at most n steps, but numerical rounding errors ensure that the method behaves more like an infinite iteration (see Reid, 1971). Moreover, for the large-scale case, we would be unwilling to consider anywhere close to n iterations. Nonetheless, the method is an effective technique for calculating approximations to the Newton direction, especially if a good preconditioner is used or if low accuracy solutions may be tolerated (see Toint, 1981, Dembo *et al.*, 1982, and Dembo and Steihaug, 1983).

In truncated-Newton methods, (see Dembo *et al.*, 1982), the method of conjugate gradients is used to generate approximations to the Newton direction. The resulting search direction, $\boldsymbol{p}_{tn}$, is employed within a linesearch framework for solving unconstrained optimization problems. Highly accurate approximations to the Newton correction are only needed to accelerate the convergence of the iteration in the neighbourhood of a limit point, and crude improvements upon the steepest-descent direction suffice elsewhere. Furthermore, by monitoring the gradient of the model at each step of the conjugate-gradient method, we can decide when to terminate the iteration.

While such an approach has undoubtedly proved successful in practice, we note that a considerable amount of work is invested, in such a scheme, in calculating an "average" direction and that much of the information gleaned on the way is subsequently ignored. We take the point of view that directions generated by the conjugate-gradient method are of interest for the quadratic model, but might also be locally of interest for the true objective function. We thus propose to construct our iterated subspace from the subspace investigated by the conjugate gradient method.

We intend to include the following:

- The preconditioned steepest-descent direction, $\boldsymbol{s}_0 = -(\boldsymbol{P}^{(k)})^{-1}\boldsymbol{g}^{(k)}$;
- A number of other conjugate directions, $\boldsymbol{s}_j$, determined by the preconditioned conjugate-gradient method; and
- The overall truncated-Newton direction, $\boldsymbol{p}_{tn}$.

We note that the first of these components is designed to encourage global convergence, while the last will ensure that convergence occurs at a fast asymptotic rate. Thus, although we could include only the steepest descent direction and not Newton or some other combination, the choice above seems most prudent.

In the next two subsections, we will discuss the choice of the conjugate directions and the size of the iterated subspace.

3.2 Choice of conjugate directions

Suppose that, in addition to the (preconditioned) steepest-descent and (truncated) Newton directions, we wish to include q directions that are $\boldsymbol{H}^{(k)}$-conjugate in the subspace. Although there are other possibilities, we propose two specific ways to choose the conjugate directions. The simplest choice is just to take the first q generated (excluding, of course, the steepest-descent direction). Alternatively one might propose choosing the q which gave the largest decrease in the model. Experiments suggest that this is rarely more successful than the simpler scheme.

Another possibility is to include approximations from the extreme eigenspaces, that is the set of eigenvectors which correspond to the smallest and largest eigenvalues (recall $\boldsymbol{H}^{(k)}$

is assumed positive definite). In the unpreconditioned case, eigenvectors corresponding to large eigenvalues correspond to the directions along which the function and CG model change most rapidly. In the preconditioned case, they still have the same effect for the transformed model.

Similarly, those associated with small eigenvalues reflect the (transformed) space in which the Newton direction is likely to be sensitive and contributions from this space are necessary if rapid progress is to be made. Thus both sets of vectors are reasonable candidates for subspace directions.

Clearly, the calculation of these spaces is generally prohibitively expensive, but they may be approximated by directions generated during the CG process (see, eg, Parlett, 1980, Chapter 13). One possibility is to monitor the Rayleigh quotients

$$\frac{\boldsymbol{s}_j^T \boldsymbol{H}^{(k)} \boldsymbol{s}_j}{\boldsymbol{s}_j^T \boldsymbol{s}_j} \tag{3.2}$$

and include the $\boldsymbol{s}_j$ which give rise to the most extreme Rayleigh quotients. Of course, these vectors are not eigenvectors of $\boldsymbol{H}^{(k)}$, but they usually contain significant contributions in the extreme eigenspaces.

3.3 Choice of subspace dimension

The choice of subspace dimension is clearly important. The simplest choice is to fix an upper bound s on this dimension before the computation proceeds (perhaps $s = 10$, see Section 4), and to select $s^{(k)}$ to be the smaller of s and the total number of CG directions sampled during the k-th CG iteration — recall that the CG process may be truncated and thus fewer than s directions may have been computed.

A more sophisticated approach is to try to dynamically select the size of subspace based upon the needs of the k-th iteration. For instance, if the Hessian is relatively well-conditioned, it is reasonable that a subspace made up from the steepest-descent and (truncated) Newton directions will suffice. If, on the other hand, the Hessian is ill-conditioned, further subspace directions to account for the extreme eigenvalues are likely to prove beneficial.

A simple heuristic would be to monitor the Rayleigh quotient as the CG iteration proceeds. Typically the first search direction, $\boldsymbol{s}_0 = -(\boldsymbol{P}^{(k)})^{-1}\boldsymbol{g}^{(k)}$, for the CG iteration will contain components of all eigenvectors and hence some components of those corresponding to the largest eigenvalues. The influence of the eigenvectors corresponding to the large eigenvalues is reduced in the subsequent directions $\boldsymbol{s}_1, \ldots$, and this is reflected in a reduction in the Rayleigh quotient during these iterations. This effect is reversed after a number of iterations, when the influence of the larger eigenvalues reappears. It would thus seem sensible to record the iteration number, $i^{(k)}$, at which the Rayleigh quotient first starts to increase after its initial sequence of decreases. As we know that we then have sampled eigenvectors corresponding to both "large" and "small" eigenvalues, it is appropriate to set $s^{(k)} = i^{(k)}$.

3.4 The inner minimization

As we have stated, we believe that there are a number of highly effective algorithms for the unconstrained minimization of a function of several variables. Indeed, it would not be unreasonable to say that the problem has effectively been solved so long as derivatives

are available. Among the most successful methods are the Newton-like second-derivative methods and the finite-difference and secant methods which require only gradients (see, for example, Dennis and Schnabel, 1983, Gill *et al.*, 1981 or Fletcher, 1987).

When considering the minimization of (1.4), we note that the calculation of derivatives of $f_s^{(k)}$ requires those of f. We see that the calculation of the second derivatives (1.6) requires significantly more products involving $\boldsymbol{S}^{(k)}$ than do the first derivatives (1.5). Thus, we would prefer to use methods which either only require relatively few Hessian-vector products, such as (preconditioned and truncated) conjugate-gradient methods, or secant methods, which build up approximations to the second derivatives from gradients in the $s^{(k)}$-dimensional subspace as they proceed.

The most widely used secant methods are those in the convex Broyden class of positive-definite approximations, of which the BFGS method has the best reputation (again see, for example, Dennis and Schnabel, 1983). While there is some controversy as to whether there are better nonconvex secant updates (see for example, Conn *et al.*, 1991, Khalfan *et al.*, 1993, and Byrd *et al.*, 1996), we feel that convex secant methods are most natural in a linesearch, as opposed to a trust-region, context. Such methods start with a positive-definite second-derivative approximation and generate a sequence of matrices which mimic the curvature in the space of directions searched. Traditionally, the Cholesky factors of the sequence are updated as the iteration proceeds. We now show that building a good starting matrix in our case is easy.

Firstly, suppose that $\boldsymbol{S}^{(k)}$ is made up purely of $\boldsymbol{H}^{(k)}$-conjugate directions. Then the exact second-derivative matrix $\boldsymbol{H}_s^{(k)}$ is diagonal because of the conjugacy and moreover its diagonal entries will have been calculated during the conjugate-gradient process. This matrix, then, is a good starting approximation; its Cholesky factors are trivial to determine. Furthermore, this choice ensures that the first quasi-Newton search direction is identical to that generated by minimizing the quadratic model (3.1) in the manifold $\boldsymbol{x}^{(k)} + \boldsymbol{S}^{(k)}\boldsymbol{y}$ (see, Gill *et al.*, 1981, section 4.8.3.1).

Now suppose that $\boldsymbol{S}^{(k)}$ is made by augmenting a set of $\boldsymbol{H}^{(k)}$-conjugate directions by the overall truncated-Newton direction $\boldsymbol{p}_{tn}$. Then, the exact second-derivative matrix has an arrowhead structure with the leading $s^{(k)} - 1$ by $s^{(k)} - 1$ submatrix being diagonal and the remaining row and column easy to obtain. To be precise, if we denote the residual $\boldsymbol{H}^{(k)}\boldsymbol{p}_{tn} + \boldsymbol{g}^{(k)}$ following the truncated conjugate-gradient process by $\boldsymbol{r}^{(k)}$, the last column of the required second-derivative matrix is $\boldsymbol{S}^{(k)T}(\boldsymbol{r}^{(k)} - \boldsymbol{g}^{(k)})$. Thus, once again this matrix provides a good starting approximation in that its Cholesky factors are extremely cheap to compute. Moreover, as before, the first quasi-Newton direction gives the minimum of the model (3.1) in the manifold $\boldsymbol{x}^{(k)} + \boldsymbol{S}^{(k)}\boldsymbol{y}$. Significantly, as the truncated-Newton direction is in the subspace, this first quasi-Newton direction is thus the same as the truncated-Newton direction.

4 Numerical Experiments

We start this section by investigating, from a numerical point of view, the impact of different subspace sizes on the convergence of the method. As the problem `FMINSURF`[1] is particularly efficiently solved using ISM in comparison with the default version of LANCELOT, we examined this problem in detail. We ran ISM, without preconditioner, but constructing the subspace from a combination of the steepest-descent, truncated-Newton and extreme

[1]From the CUTE collection, see Bongartz *et al.* (1995).

directions as described in Section 3.2, using a variety of subspace dimensions and illustrate, in Figure 1, the effects of the choice of this dimension on the CPU time[2] required to solve the problem.

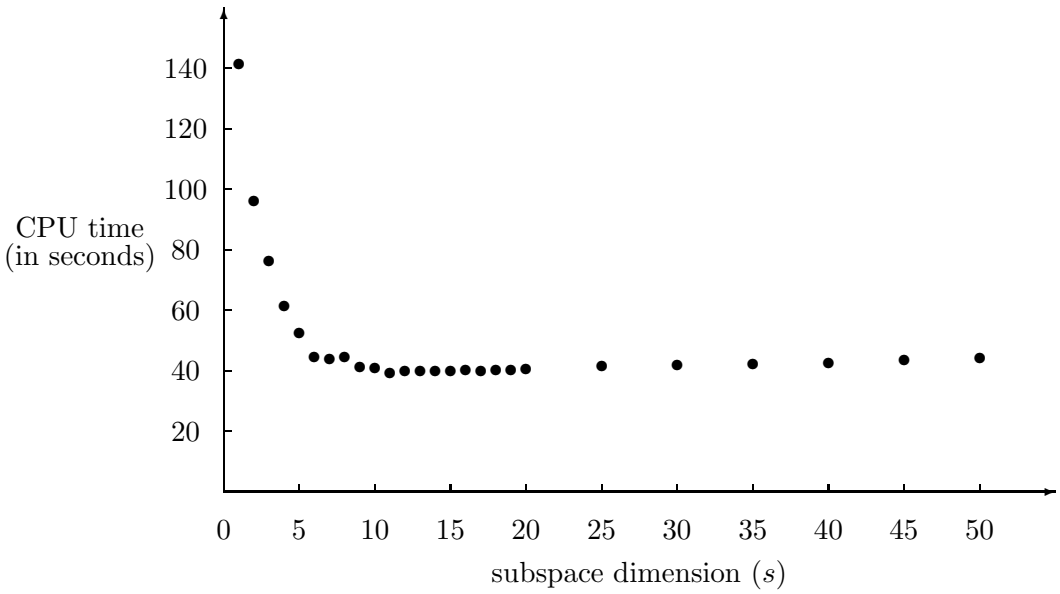

Figure 1: The impact of varying the subspace dimension on `FMINSURF` (using an unpreconditioned ISM in which the subspace is chosen from the the steepest-descent, truncated-Newton and extreme CG directions).

We observe that the CPU time for small subspace dimension is large, but that, for subspaces of dimension between nine and forty, the time is relatively constant, being within ten percent of the least ($s = 11$) time. Thus, it appears that for this problem, adding information above the steepest-descent and truncated-Newton directions is beneficial but there is little extra payoff from using more than eleven directions. When we monitored the Rayleigh quotients for this problem, we observed that the quotient decreases for on average ten CG iterations before increasing for the first time. Thus, unless we insist on at least ten CG iterations, it is possible that we may not have sampled the complete eigenspace. Similar runs on different problems indicate that this behaviour is not exceptional. In other words it appears to be not exceptional that searching in a subspace that uses eight directions corresponding to the extreme Rayleigh quotients with the addition of the steepest descent and the truncated Newton directions captures the bulk of the possible improvements at a given iteration. Of course, this cannot be the case for all iterations or all problems, but the overhead associated with choosing substantially more directions, and the loss of descent incurred by using less directions (recall s is fixed), does not payoff in general.

We next report the results of running a number of variants of our Iterated-Subspace

[2]This experiment was performed on an IBM RISC/6000 320h workstation, using optimized (-O) Fortran 77 code and IBM-supplied BLAS.

Minimization code, based on the suggestions made in Section 3, on some large or difficult test problems. The simplest variant has the following features.

- *No* preconditioning is used.
- $s^{(k)}$ is the smaller of the *fixed* bound $s = 10$ (as suggested by the above example) and the number of inner iterations required to determine the truncated-Newton direction.
- The subspace is constructed from the *first* $s^{(k)} - 1$ conjugate directions plus the truncated-Newton direction. The truncation is performed when the residual (gradient) of the model is smaller than $\|\boldsymbol{g}^{(k)}\|_2 \min(0.1, \|\boldsymbol{g}^{(k)}\|_2^{0.1})$ or when more than n conjugate-gradient iterations were performed.
- The model is modified, if necessary, to ensure that it is strictly convex. The modification is carried out as the conjugate-gradient iteration proceeds using the method of Arioli *et al.* (1993).
- The iteration is deemed to have converged when $\|\boldsymbol{g}^{(k)}\|_2$ is smaller than 10^{-5}.
- A BFGS linesearch method is used to solve the inner-minimization problem. An Armijo backtracking linesearch is used, starting with a unit step and dividing the step by two until the Armijo sufficient decrease condition is satisfied. If a step of one proves acceptable, but the model has been modified to ensure that it is strictly convex, the step is doubled until an unacceptable stepsize is determined, when the last-found acceptable step is chosen. A maximum of $2s^{(k)}$ BFGS iterations are permitted and the iteration is stopped if $\|\boldsymbol{g}_s^{(k)}\|_2$ is smaller than 10^{-6}.

Note that one Hessian evaluation is made for each major iteration and one gradient evaluation for each inner iteration. We denote this method by the symbol ISM(n,f,f), where n means that *no* preconditioning is used, the first f that the subspace is constructed from the *first* CG directions, and the second f that the maximal subspace dimension s is *fixed* (to ten).

We also consider the ISM(p,f,f) method, which is identical to ISM(n,f,f), except that an 11-band modified Cholesky factorization *preconditioner* is used in the conjugate-gradient calculation. The factorization takes the elements of $\boldsymbol{H}^{(k)}$ within a band of semi-bandwidth five of the diagonal, replacing any other elements by zeros. The resulting band matrix is factorized, modifications being made according to the method of Schnabel and Eskow (1991) to ensure that the preconditioner is positive definite with bounded condition number.

We define the methods ISM(n,e,f) and ISM(p,e,f) as the modifications of ISM(n,f,f) and ISM(p,f,f), respectively, where we construct the subspace from a set of *extreme* $s^{(k)} - 2$ conjugate directions plus the steepest-descent and truncated-Newton directions. We pick half of the extreme directions to be those whose Rayleigh quotient is largest, while the remainder correspond to the smallest Rayleigh quotients.

We next consider the possibility of generating the subspace dimension *automatically*, as discussed in Section 3.3. Once the subspace dimension has been determined, we construct the subspace from the set of first or extreme $s^{(k)} - 2$ conjugate directions plus the steepest-descent and truncated-Newton direction, just as before. This results in four additional variants, namely ISM(n,f,a), ISM(p,f,a), ISM(n,e,a) and ISM(p,e,a).

As a yard-stick, we compare the above methods with three other algorithms. The first is the default version of the LANCELOT A nonlinear optimization package (see Conn *et*

al., 1992) (denoted LAN(p)) in which an 11-band preconditioner is used, together with the same unpreconditioned version (denoted LAN(n)). LANCELOT is a trust-region method in which a gradient and Hessian evaluation are made on every successful iteration. The trust region subproblem is solved using a truncated conjugate gradient method. The second algorithm included in our comparison is a truncated-Newton method (see Dembo and Steihaug (1983)). The truncated-Newton search direction is obtained by an inexact minimization of the Newton model using unpreconditioned or preconditioned conjugate gradients. We denote the resulting methods by TN(n) and TN(p), respectively. These two variants are obtained from our ISM algorithms by restricting the subspace minimization to a single linesearch along the truncated-Newton direction. Here, the truncation is performed when the residual (gradient) of the model is smaller than $\|\boldsymbol{g}^{(k)}\|_2 \min(0.1, \|\boldsymbol{g}^{(k)}\|_2^{0.5})$ or when

more than n conjugate-gradient iterations are performed[3]. Finally, we also compare our ISM algorithms with the limited memory algorithm L-BFGS-B of Zhu *et al.* (1996) (see also Byrd *et al.*, 1995). In the unconstrained case, Algorithm L-BFGS-B approximately minimizes a quadratic model, whose Hessian is a limited memory BFGS approximation of the Hessian of the objective function, to compute a search direction, and performs a linesearch along this search direction. We set parameter **isbmin** to three in the code, meaning that the conjugate-gradient method is used to compute the search direction. As no preconditioning is considered in this method (denoted LM(n)), we compare it with the unpreconditioned variants of the ISM algorithms. We note that all the algorithms considered in this comparison, except the limited memory one of course, use exact first and second derivatives.

We selected our 34 test examples as the majority of large and/or difficult unconstrained test examples in the CUTE (see Bongartz *et al.*, 1995) test set. Only problems which took excessive CPU time (more than 30 minutes), or which were variations on the reported problems, were excluded. All experiments were made on a DEC 3000 workstation, using optimized (-O) Fortran 77 code.

Table 1: Cumulative statistics on the performance of all methods on 32 problems (unpreconditioned case)

Method	Details	#f	Time
LAN(n)	Table 5	5585	486
TN(n)	Table 7	19443	649
LM(n)	Table 9	15046	1175
ISM(n,f,f)	Table 10	21743	695
ISM(n,e,f)	Table 12	22168	720
ISM(n,f,a)	Table 14	14753	532
ISM(n,e,a)	Table 16	15172	533

Table 2: Cumulative statistics on the performance of all except the limited memory method on 32 problems (preconditioned case)

Method	Details	#f	Time
LAN(p)	Table 6	4197	456
TN(p)	Table 8	7852	471
ISM(p,f,f)	Table 11	22096	649
ISM(p,e,f)	Table 13	22193	652
ISM(p,f,a)	Table 15	12616	448
ISM(p,e,a)	Table 17	12774	458

Tables 1 and 2 first report cumulative statistics on the performance of the considered algorithms on 32 problems for the unpreconditioned case (the limited memory algorithm

[3]The exponent differs from the 0.1 used for the ISM methods because otherwise the results for the truncated-Newton method deteriorate considerably.

failed to solve problems DIXON3DQ and VARDIM), and on 32 problems for the preconditioned case (different approximate local optima have been reached for problems LIARWHD and NONDIA). In these tables and the following ones, $\#f$ indicates the total number of function evaluations and *time* the total CPU time (in seconds). The second column indicates which of the tables in the appendix gives the complete and detailed results for the considered method.

These tables show some interesting results. In particular, they indicate that the automatic choice of the subspace dimension is advantageous on average. On the other hand, there is little difference between the average performance of ISM methods using the first CG directions to define the subspace and those using the extreme ones. One also sees that, on average, the preconditioned ISM variants are all better in CPU time than their unpreconditioned counterparts. When comparing with the other methods, the unpreconditioned ISM variants with automatic choice of the subspace dimension require, on average, more function evaluations than LANCELOT, but approximately the same amount as limited memory and less than truncated-Newton. In the preconditioned case, the total number of function evaluations is less for LANCELOT and truncated-Newton. The overall CPU time for the unpreconditioned case indicates a better performance, on average, for LANCELOT, directly followed (within less than ten per cent) by ISM variants with automatic choice of the subspace dimension. The limited memory method did not perform as well on these examples. This is more than likely because it uses less derivative information. When preconditioning is used, the total CPU time for the ISM methods with automatic choice of the subspace dimension is comparable with (and even slightly better than) the total CPU time for LANCELOT and for truncated-Newton.

We next consider a more disaggregate presentation of what happens problem by problem to refine our analysis. We present in Tables 3 and 4 the number of times that each of the considered method ranks first, second, third, etc. for the two criteria used above. In these rankings, two CPU times are reputed identical if they differ by less than five percent or by less than half a second.

Table 3: Rankings (unpreconditioned case)

Method	Function evaluations (#f)							Time						
	1st	2nd	3rd	4th	5th	6th	7th	1st	2nd	3rd	4th	5th	6th	7th
LAN(n)	26	2	1	1	0	2	0	12	3	1	1	3	6	6
TN(n)	3	9	4	3	0	0	13	18	0	0	1	1	0	12
LM(n)	1	10	1	2	4	5	9	14	1	1	0	5	5	6
ISM(n,f,f)	1	6	6	3	5	9	2	19	0	1	3	4	2	3
ISM(n,e,f)	1	6	5	3	5	8	4	19	1	1	1	2	7	1
ISM(n,f,a)	3	9	7	9	4	0	0	22	1	2	4	1	2	0
ISM(n,e,a)	3	6	9	8	4	2	0	21	0	4	2	4	1	0

These tables strengthen the indications drawn above from average measures, except that they indicate that the supremacy of LANCELOT in CPU time for the unpreconditioned ISM variants is misleading. It appears indeed that all the ISM methods (with or without preconditioner) are very competitive in CPU time. We further observe that truncated-Newton is sometimes very good and sometimes rather poor for the unpreconditioned case

Table 4: Rankings (preconditioned case)

Method	Function evaluations (#f)						Time					
	1st	2nd	3rd	4th	5th	6th	1st	2nd	3rd	4th	5th	6th
LAN(p)	31	0	0	0	0	1	19	2	0	4	0	7
TN(p)	5	13	0	10	0	4	20	0	0	2	3	7
ISM(p,f,f)	5	2	3	5	16	1	17	2	1	3	7	2
ISM(p,e,f)	5	2	4	3	15	3	15	0	3	3	9	2
ISM(p,f,a)	5	13	9	1	3	1	19	4	6	2	0	1
ISM(p,e,a)	6	8	11	2	5	0	19	2	4	4	3	0

(as may be expected), while this dichotomy is less marked in the preconditioned case.

We thus conclude that in general ISM methods are efficient from the CPU time point of view, even though they may require substantially more function evaluations than LANCELOT (but not more than truncated-Newton and limited memory). Further, the automatic determination of the subspace dimension is often quite helpful, both in the unpreconditioned and the preconditioned cases. On the other hand, there is little difference in performance between ISM variants that build the subspace from the first CG directions and variants that use the extreme ones. As the former is easier to implement, one might prefer it in practice. In fact ISM(p,f,a) is the best, slightly[4].

5 Perspectives and Conclusions

In this paper, we have shown that it is possible to solve large-scale nonlinear optimization problems using methods designed for small-scale problems. These methods may be regarded as a generalization of linesearch-type methods more usually used to solve unconstrained minimization problems. We have indicated that the convergence of our methods depends upon using robust algorithms for the small-dimensional subproblems, and have suggested a number of ways of selecting promising subspaces in which to search. Furthermore, we feel that there are a number of important areas for future investigation.

- It is possible to extend the iterated-subspace minimization idea to treat linearly constrained optimization problems. For example, if we suppose the original problem is of the form

$$\underset{\boldsymbol{x}\in\Re^n}{\text{minimize}} \quad f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{l} \leq \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{u}, \tag{5.1}$$

  where $\boldsymbol{A}$ is an m by n matrix, $\boldsymbol{l}$ and $\boldsymbol{u}$ are m-vectors and $\boldsymbol{x}^{(k)}$ satisfies $\boldsymbol{l} \leq \boldsymbol{A}\boldsymbol{x}^{(k)} \leq \boldsymbol{u}$, we may apply the following linearly-constrained ISM algorithm:

  1. Stop if convergence tests are satisfied.
  2. Determine a full-rank subspace matrix $\boldsymbol{S}^{(k)} \in \Re^{n\times s^{(k)}}$, where $s^{(k)} \ll n$.

[4] One should also bear in mind, at this point, that LANCELOT is a much more sophisticated code than our ISM variants, because it is designed for solving generally constrained in addition to unconstrained problems, has been extensively tested and refined, and contains a number of safeguards that are not included in the simpler ISM codes.

3. Approximately solve the $s^{(k)}$-dimensional minimization problem

$$\underset{\boldsymbol{y}\in\Re^{s^{(k)}}}{\text{minimize}} \quad f(\boldsymbol{x}^{(k)}+\boldsymbol{S}^{(k)}\boldsymbol{y}) \quad \text{subject to} \quad \boldsymbol{l}^{(k)} \leq \boldsymbol{A}^{(k)}\boldsymbol{y} \leq \boldsymbol{u}^{(k)}, \tag{5.2}$$

where $\boldsymbol{A}^{(k)} = \boldsymbol{A}\boldsymbol{S}^{(k)}$, $\boldsymbol{l}^{(k)} = \boldsymbol{l} - \boldsymbol{A}\boldsymbol{x}^{(k)}$ and $\boldsymbol{u}^{(k)} = \boldsymbol{u} - \boldsymbol{A}\boldsymbol{x}^{(k)}$, and set

$$\boldsymbol{x}^{(k+1)} = \text{(approximate) arg min}_{\boldsymbol{y}\in\Re^{s^{(k)}}} \quad f(\boldsymbol{x}^{(k)}+\boldsymbol{S}^{(k)}\boldsymbol{y}) \tag{5.3}$$

$$\text{subject to} \quad \boldsymbol{l}^{(k)} \leq \boldsymbol{A}^{(k)}\boldsymbol{y} \leq \boldsymbol{u}^{(k)}.$$

The central issues remain those discussed in Section 1. However, extra care must be exercised when picking the subspace matrix, as it is now desirable for a *constrained* steepest-descent and (truncated) Newton directions to lie in the subspace. The real difficulty is in adding appropriate additional directions since, for example, a straightforward analogue of the unconstrained case would require that we have conjugate directions in the constrained (reduced) space. One instance where this supposition is reasonable is when the constraints are just simple bounds. We also now need to use efficient methods for solving *small linearly-constrained minimization* problems when determining $\boldsymbol{x}^{(k+1)}$, but fortunately the state-of-the-art here is as advanced as it is for unconstrained minimization.

The ISM philosophy does not obviously extend to handle nonlinearly-constrained minimization problems except that, of course, any unconstrained or linearly-constrained subproblems may be treated by existing ISM methods. This may be important for nonlinearly-constrained minimization methods which are based on the sequential minimization of penalty or barrier functions, or their augmented or shifted counterparts.

- In our investigations, we found it convenient to use a linesearch (BFGS) method to solve the inner-iteration subproblems. One might, of course, alternatively use a trust-region method to solve the subproblem. However, as the performance of such methods depends upon building a good model within an adequate trust region, and as our ISM method will solve a sequence of subproblems, it may be that a good trust-region radius for one subproblem is poor for the next, and inefficiencies may occur. Thus care may be needed in determining interactions between successive models.

  Another important issue is how to pick stopping rules, analogous to (2.5)–(2.7), which are appropriate for trust-region based methods. The main difficulty here is that the initial trust-region radius may interfere with a condition like (2.5).

- While we have suggested a number of methods for computing a good iterated subspace, more work clearly needs to be performed. We believe that we have identified some of the ingredients of a good subspace, but our understanding is far from complete.

6 Acknowledgement

Nick Gould would like to thank CERFACS for the facilities which made some of this research possible.

References

[Arioli *et al.*, 1993] M. Arioli, T. F. Chan, I. S. Duff, N. I. M. Gould, and J. K. Reid. Computing a search direction for large-scale linearly constrained nonlinear optimization calculations. Technical Report Research Report 93-066, RAL, Chilton, England, 1993.

[Armijo, 1966] L. Armijo. Minimization of functions having Lipschitz-continuous first partial derivatives. *Pacific Journal of Mathematics*, 16:1–3, 1966.

[Bertsekas, 1982] D. P. Bertsekas. *Constrained Optimization and Lagrange Multipliers Methods*. Academic Press, London, 1982.

[Bongartz *et al.*, 1995] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: Constrained and unconstrained testing environment. *ACM Transactions on Mathematical Software*, 21(1):123 – 160, 1995.

[Byrd *et al.*, 1995] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[Byrd *et al.*, 1996] R. H. Byrd, H.F. Khalfan, and R. B. Schnabel. Analysis of a symmetric rank-¡one trust region method. *SIAM Journal on Optimization*, 1996. to appear.

[Conn *et al.*, 1991] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Convergence of quasi-Newton matrices generated by the symmetric rank one update. *Mathematical Programming*, 50(2):177–196, 1991.

[Conn *et al.*, 1992] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. LANCELOT*: a Fortran package for large-scale nonlinear optimization (Release A)*, volume 17 of *Springer Series in Computational Mathematics*. Springer Verlag, Heidelberg, Berlin, New York, 1992.

[Conn *et al.*, 1993] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Complete numerical results for LANCELOT Release A. Research Report RC 18750, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA, 1993. Also issued as Technical Report 92/15, Department of Mathematics, FUNDP, Namur, Belgium.

[Conn *et al.*, 1996] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization. *Mathematical Programming*, to appear, 1996.

[Cragg and Levy, 1969] E. E. Cragg and A. V. Levy. Study on a supermemory gradient method for the minimization of functions. *Journal of Optimization Theory and Applications*, 4(3):191–205, 1969.

[Dembo and Steihaug, 1983] R. S. Dembo and T. Steihaug. Truncated-Newton algorithms for large-scale unconstrained optimization. *Mathematical Programming*, 26:190–212, 1983.

[Dembo *et al.*, 1982] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact-Newton methods. *SIAM Journal on Numerical Analysis*, 19(2):400–408, 1982.

[Dennis and Schnabel, 1983] J. E. Dennis and R. B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. Prentice-Hall, Englewood Cliffs, NJ, 1983.

[Dennis and Turner, 1987] J. E. Dennis and K. Turner. Generalized conjugate directions. *Linear Algebra and Applications*, 88/89:187–209, 1987.

[Dixon *et al.*, 1984] L. C. W. Dixon, K. D. Patel, and P. G. Ducksbury. Experience running optimization algorithms on parallel processing system. In *11th IFIP Conference on System Modelling and Optimization*, volume 59, pages 891–932, Berlin, 1984. Springer Verlag. Lecture Notes in Control and Information Sciences.

[Fletcher, 1987] R. Fletcher. *Practical Methods of Optimization.* J. Wiley and Sons, Chichester, second edition, 1987.

[Gill *et al.*, 1981] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, London and New York, 1981.

[Goldstein, 1964] A.A. Goldstein. Convex programming in Hilbert space. *Bulletin of the American Mathematical Society*, 70:709–710, 1964.

[Golub and Loan, 1989] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, second edition, 1989.

[Hestenes and Stiefel, 1952] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. N.B.S.*, 49:409–436, 1952.

[Kaporin and Axelsson, 1995] I. E. Kaporin and O. Axelsson. On a class of nonlinear equation solvers based on the residual norm reduction over a sequence of affine subspaces. *SIAM Journal on Scientific Computing*, 16(1):228–249, 1995.

[Khalfan *et al.*, 1993] H. Khalfan, R. H. Byrd, and R. B. Schnabel. The symmetric rank one update. Technical Report CU-CS-240, Department of Computer Science, University of Colorado, Boulder, USA", 1993.

[Miele and Cantrell, 1969] A. Miele and J. W. Cantrell. Study on a memory gradient method for the minimization of functions. *Journal of Optimization Theory and Applications*, 3:459–470, 1969.

[Parlett, 1980] B. N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, Englewood Cliffs, USA, 1980.

[Powell, 1970] M. J. D. Powell. A hybrid method for nonlinear equations. In P. Rabinowitz, editor, *Numerical Method for Nonlinear Algebraic Equations*, pages 87–114. Gordon and Breach, London, 1970.

[Reid, 1971] J. K. Reid. On the method of conjugate gradients for the solution large sparse linear equations. In J. K. Reid, editor, *Large sparse sets of linear equations*, pages 231–254, London, 1971. Academic Press.

[Saad, 1990] Y. Saad. Krylov subspace methods: theory, algorithms and applications. In *Computational Methods in Applied Science and Engineering, INRIA 29 January - 2 February.* INRIA, 1990.

[Schnabel and Eskow, 1991] R. B. Schnabel and E. Eskow. A new modified Cholesky factorization. *SIAM Journal on Scientific and Statistical Computing*, 11:1136–1158, 1991.

[Toint, 1981] Ph. L. Toint. Towards an efficient sparsity exploiting Newton method for minimization. In I. S. Duff, editor, *Sparse Matrices and Their Uses*, pages 57–88. Academic Press, London, 1981.

[Vinsome, 1976] P. K. W. Vinsome. Orthomin, an iterative method for solving sparse sets of simultaneous linear equations. In *Proceedings of the Fourth Symposium on Reservoir Simulation*. Society of Petroleum Engineers of AIME, 1976.

[Zhu *et al.*, 1996] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization. *ACM Transactions on Mathematical Software*, (to appear), 1996.

A Detailed numerical results

In this appendix, we give comprehensive details of the performance of each of the methods discussed in the main body of the paper on the complete set of test examples. We include these results so that others may, in future, compare new proposals with ours.

Table 5: Results for LAN(n) (unpreconditioned LANCELOT) on large or hard problems.

problem	n	#f	#it	#cg	time	f
ARWHEAD	1000	6	5	2	0.95	1.6903D-10
BDQRTIC	1000	13	12	66	2.01	3.9838D+03
BROWNAL	100	3	2	3	0.17	1.1263D-13
BRYBND	1000	14	13	177	3.53	2.8797D-13
CRAGGLVY	1000	14	13	166	2.40	3.3642D+02
DIXMAANA	1500	7	6	10	1.40	1.0000D+00
DIXON3DQ	1000	6	5	1427	5.04	7.7032D-09
DQDRTIC	1000	3	2	0	0.73	1.6602D-23
DQRTIC	1000	36	35	223	2.87	3.7687D-06
EDENSCH	1000	15	14	35	1.58	6.0033D+03
EIGENALS	110	16	15	142	0.76	2.7564D-11
ENGVAL1	1000	8	7	20	1.08	1.1082D+03
FLETCHCR	1000	3529	3528	20619	219.85	5.0581D-14
FMINSURF	1024	190	189	527	15.69	1.0000D+00
FREUROTH	1000	21	20	40	1.76	1.2147D+05
GENROSE	1000	1214	1213	6640	77.43	1.0000D+00
LIARWHD	1000	14	13	14	1.27	2.3011D-16
MANCINO	100	12	11	13	11.28	4.3367D-17
MOREBV	1000	3	2	265	2.01	2.0186D-09
NCB20B	1000	28	27	1274	96.53	1.6760D+03
NONDIA	1000	30	29	29	1.99	4.3483D-16
NONDQUAR	1000	92	91	853	6.43	9.5606D-06
PENALTY1	1000	56	55	44	8.15	9.6862D-03
POWELLSG	1000	16	15	60	1.35	1.1167D-06
POWER	1000	29	28	613	3.73	5.8399D-09
QUARTC	1000	36	35	223	3.07	3.7687D-06
SINQUAD	1000	70	69	126	5.54	6.3238D-05
SROSENBR	1000	11	10	20	1.06	6.6723D-12
TOINTGSS	1000	9	8	15	0.94	1.0010D+01
TQUARTIC	1000	14	13	14	1.26	1.0168D-16
TRIDIA	1000	5	4	91	1.14	8.5487D-14
VARDIM	1000	37	36	0	1.87	1.1203D-20
VAREIGVL	1000	13	12	313	4.32	3.0691D-08
WOODS	1000	58	57	194	3.42	3.5419D-16

Key: n = number of variables, $\#f$ = number of function evaluations, $\#it$ = number of iterations, $\#cg$ = total number of CG iterations, *time* = total CPU time in seconds, f = smallest function value obtained.

Table 6: Results for LAN(p) (default preconditioned LANCELOT) on large or hard problems.

problem	n	#f	#it	#cg	time	f
ARWHEAD	1000	6	5	1	0.72	1.6903D-10
BDQRTIC	1000	12	11	13	1.56	3.9838D+03
BROWNAL	100	4	3	40	0.97	6.3470D-11
BRYBND	1000	17	16	30	2.53	1.8775D-12
CRAGGLVY	1000	15	14	11	1.51	3.3642D+02
DIXMAANA	1500	8	7	10	1.14	1.0000D+00
DIXON3DQ	1000	3	2	2	0.40	0.0000D+00
DQDRTIC	1000	3	2	0	0.40	1.6602D-23
DQRTIC	1000	36	35	27	2.31	3.6952D-06
EDENSCH	1000	13	12	9	1.24	6.0033D+03
EIGENALS	110	25	24	56	1.18	1.4887D-12
ENGVAL1	1000	8	7	7	0.81	1.1082D+03
FLETCHCR	1000	2138	2137	2136	119.01	2.8160D-12
FMINSURF	1024	316	315	436	106.32	1.0000D+00
FREUROTH	1000	11	10	7	1.06	1.2147D+05
GENROSE	1000	1095	1094	1158	63.37	1.0000D+00
LIARWHD	1000	15	14	21	1.29	2.0274D-20
MANCINO	100	16	15	8	17.39	9.2449D-18
MOREBV	1000	2	1	1	0.39	7.3289D-13
NCB20B	1000	23	22	568	50.35	1.6760D+03
NONDIA	1000	30	29	47	2.18	6.9910D-16
NONDQUAR	1000	18	17	19	1.21	1.3932D-09
PENALTY1	1000	64	63	432	23.16	9.6862D-03
POWELLSG	1000	16	15	15	1.06	2.2324D-06
POWER	1000	28	27	53	7.60	1.8888D-08
QUARTC	1000	36	35	27	2.32	3.6952D-06
SINQUAD	1000	132	131	341	14.82	9.0365D-07
SROSENBR	1000	11	10	10	0.82	5.8309D-13
TOINTGSS	1000	3	2	2	0.44	1.0000D+01
TQUARTIC	1000	13	12	24	1.19	2.9487D-11
TRIDIA	1000	3	2	1	0.49	1.3827D-32
VARDIM	1000	37	36	0	19.03	1.1203D-20
VAREIGVL	1000	13	12	199	7.06	4.1621D-08
WOODS	1000	72	71	69	4.18	2.4231D-11

Key: n = number of variables, $\#f$ = number of function evaluations, $\#it$ = number of iterations, $\#cg$ = total number of CG iterations, *time* = total CPU time in seconds, f = smallest function value obtained.

Table 7: Results for the unpreconditioned truncated-Newton method TN(n) on large or hard problems.

problem	n	#f	#its	#cg	time	f
ARWHEAD	1000	1545	389	395	26.63	0.00D+00
BDQRTIC	1000	43	17	97	1.68	3.98D+03
BROWNAL	100	5	3	4	0.04	2.45D-08
BRYBND	1000	37	14	48	1.49	1.22D-12
CRAGGLVY	1000	45	16	129	1.98	3.36D+02
DIXMAANA	1500	189	64	73	7.10	1.00D+00
DIXON3DQ	1000	7	4	1748	3.91	1.25D-11
DQDRTIC	1000	10	5	12	0.23	1.91D-14
DQRTIC	1000	48	17	70	0.38	5.14D-08
EDENSCH	1000	46	17	44	1.35	6.00D+03
EIGENALS	110	69	22	179	0.98	8.43D-12
ENGVAL1	1000	31	13	28	0.81	1.11D+03
FLETCHCR	1000	3046	1184	13315	100.03	3.04D-15
FMINSURF	1024	156	26	5559	46.24	1.00D+00
FREUROTH	1000	92	19	110	2.42	1.21D+05
GENROSE	1000	2896	540	9838	71.28	1.00D+00
LIARWHD	1000	78	28	32	1.42	1.43D-16
MANCINO	100	72	21	29	68.27	1.87D-21
MOREBV	1000	3	2	549	2.14	1.34D-09
NCB20B	1000	74	19	1053	81.98	1.68D+03
NONDIA	1000	156	55	58	3.46	5.93D-21
NONDQUAR	1000	382	94	26963	67.04	3.76D-08
PENALTY1	1000	124	52	78	1.73	9.69D-03
POWELLSG	1000	837	281	575	6.04	5.31D-08
POWER	1000	5636	1414	1419	29.24	4.19D-09
QUARTC	1000	48	17	70	0.38	5.14D-08
SINQUAD	1000	285	119	216	9.06	1.33D-06
SROSENBR	1000	26	12	15	0.29	6.11D-20
TOINTGSS	1000	298	101	202	8.52	1.00D+01
TQUARTIC	1000	57	20	27	0.74	1.24D-13
TRIDIA	1000	16	9	576	1.40	1.31D-14
VARDIM	1000	60	15	15	0.33	2.28D-21
VAREIGVL	1000	3018	1007	1010	103.70	6.02D-11
WOODS	1000	75	24	63	0.96	6.63D-18

Key: n = number of variables, $\#f$ = number of function evaluations, $\#its$ = total number of iterations, $\#cg$ = total number of CG iterations, *time* = total CPU time in seconds, f = smallest function value obtained.

Table 8: Results for the preconditioned truncated-Newton method TN(p) on large or hard problems.

problem	n	#f	#its	#cg	time	f
ARWHEAD	1000	20	8	10	0.68	0.00D+00
BDQRTIC	1000	30	11	15	1.17	3.98D+03
BROWNAL	100	74	26	213	7.10	1.21D-11
BRYBND	1000	72	15	41	2.65	1.91D-16
CRAGGLVY	1000	122	42	42	4.91	3.36D+02
DIXMAANA	1500	91	31	35	4.56	1.00D+00
DIXON3DQ	1000	3	1	1	0.05	0.00D+00
DQDRTIC	1000	3	1	1	0.09	0.00D+00
DQRTIC	1000	43	15	15	0.45	7.86D-08
EDENSCH	1000	42	12	13	1.30	6.00D+03
EIGENALS	110	140	36	162	2.19	8.15D-12
ENGVAL1	1000	26	10	10	0.82	1.11D+03
FLETCHCR	1000	2962	1169	1172	79.57	4.78D-14
FMINSURF	1024	248	28	1438	25.48	1.00D+00
FREUROTH	1000	47	9	1008	19.38	1.21D+05
GENROSE	1000	2290	748	1548	58.31	1.00D+00
LIARWHD	1000	48	16	27	1.24	3.51D-15
MANCINO	100	105	23	55	88.31	3.43D-21
MOREBV	1000	2	1	1	0.08	7.33D-13
NCB20B	1000	47	16	565	49.92	1.68D+03
NONDIA	1000	50	19	35	1.57	8.47D-20
NONDQUAR	1000	20	9	17	0.39	2.45D-09
PENALTY1	1000	106	32	223	9.71	9.69D-03
POWELLSG	1000	21	10	16	0.37	5.55D-08
POWER	1000	43	15	15	3.81	1.00D-10
QUARTC	1000	43	15	15	0.45	7.86D-08
SINQUAD	1000	784	307	861	37.85	2.35D-10
SROSENBR	1000	36	12	13	0.53	8.11D-25
TOINTGSS	1000	3	1	1	0.14	1.00D+01
TQUARTIC	1000	25	9	16	0.55	6.46D-21
TRIDIA	1000	3	1	1	0.05	1.97D-26
VARDIM	1000	78	28	3241	29.14	9.73D-16
VAREIGVL	1000	179	49	1932	38.61	2.35D-14
WOODS	1000	144	35	48	2.11	3.78D-15

Key: n = number of variables, $\#f$ = number of function evaluations, $\#its$ = total number of iterations, $\#cg$ = total number of CG iterations, *time* = total CPU time in seconds, f = smallest function value obtained.

Table 9: Results for the (unpreconditioned) limited memory method LM(n) on large or hard problems.

problem	n	#f	#its	#cg	time	f
ARWHEAD	1000	22	13	20	0.70	0.000D+00
BDQRTIC	1000	294	196	1104	11.84	3.984D+03
BROWNAL	100	22	9	18	0.12	2.449D-08
BRYBND	1000	31	24	80	1.39	2.152D-11
CRAGGLVY	1000	94	81	467	5.50	3.364D+02
DIXMAANA	1500	10	7	10	0.58	1.000D+00
DIXON3DQ	1000	9999	9668		343.62	5.065D-05
DQDRTIC	1000	18	11	31	0.52	9.088D-14
DQRTIC	1000	48	38	73	0.86	8.112D-06
EDENSCH	1000	31	23	77	1.29	6.003D+03
EIGENALS	110	444	408	2565	6.30	1.609D-09
ENGVAL1	1000	24	17	52	0.87	1.108D+03
FLETCHCR	1000	5795	5108	35941	255.57	7.962D-12
FMINSURF	1024	186	175	1219	11.29	1.000D+00
FREUROTH	1000	114	29	91	4.25	1.215D+05
GENROSE	1000	2460	2163	14874	111.52	1.000D+00
LIARWHD	1000	28	22	40	0.88	2.268D-16
MANCINO	100	11	9	11	15.46	8.786D-20
MOREBV	1000	75	71	526	3.32	9.964D-09
NCB20B	1000	2581	2393	18491	651.93	1.676D+03
NONDIA	1000	28	17	30	0.99	7.125D-21
NONDQUAR	1000	1073	961	7250	35.67	1.521D-05
PENALTY1	1000	121	85	166	2.76	9.687D-03
POWELLSG	1000	48	37	116	0.99	6.583D-06
POWER	1000	149	131	773	4.11	1.218D-08
QUARTC	1000	48	38	73	0.84	8.112D-06
SINQUAD	1000	141	104	385	6.41	1.129D-04
SROSENBR	1000	21	15	24	0.41	1.037D-09
TOINTGSS	1000	8	5	9	0.36	1.001D+01
TQUARTIC	1000	25	19	34	0.62	4.907D-17
TRIDIA	1000	912	871	5992	29.15	1.606D-11
VARDIM	1000	21	1	0	0.17	1.242D+22
VAREIGVL	1000	107	99	646	6.97	9.973D-09
WOODS	1000	77	52	178	2.01	1.287D-11

Key: n = number of variables, $\#f$ = number of function evaluations, $\#its$ = total number of iterations, $\#cg$ = total number of CG iterations, *time* = total CPU time in seconds, f = smallest function value obtained.

Table 10: Results for ISM(n,f,f) on large or hard problems, in which no preconditioning is used and the subspace is chosen from the first 10 CG directions.

problem	n	#f	#min	#its	#cg	$s^{(k)}$	time	f
ARWHEAD	1000	11	2	6	3	1.50	0.25	0.00D+00
BDQRTIC	1000	85	10	47	63	5.40	2.12	3.98D+03
BROWNAL	100	5	2	3	3	1.50	0.04	2.45D-08
BRYBND	1000	105	10	80	60	5.90	3.62	8.79D-14
CRAGGLVY	1000	108	12	90	126	7.58	4.46	3.36D+02
DIXMAANA	1500	24	7	15	7	1.00	1.09	1.00D+00
DIXON3DQ	1000	7	4	4	1748	9.00	3.97	1.25D-11
DQDRTIC	1000	10	5	5	12	2.40	0.24	1.91D-14
DQRTIC	1000	188	8	145	21	2.63	1.18	9.10D-09
EDENSCH	1000	65	7	44	25	3.57	1.76	6.00D+03
EIGENALS	110	101	8	64	86	9.00	1.16	1.90D-10
ENGVAL1	1000	52	7	34	22	3.14	1.19	1.11D+03
FLETCHCR	1000	13014	1049	10640	13187	9.92	316.24	3.60D-14
FMINSURF	1024	189	11	103	2309	10.00	21.61	1.00D+00
FREUROTH	1000	261	11	69	47	4.09	4.92	1.21D+05
GENROSE	1000	5985	411	4327	6827	9.90	138.24	1.00D+00
LIARWHD	1000	29	3	24	4	1.33	0.57	1.46D-14
MANCINO	100	68	12	37	17	1.42	67.72	1.34D-18
MOREBV	1000	4	3	3	519	10.00	1.98	1.37D-09
NCB20B	1000	174	17	101	1034	9.88	99.47	1.68D+03
NONDIA	1000	14	3	10	4	1.33	0.38	1.48D-15
NONDQUAR	1000	466	40	368	2643	9.05	10.84	2.11D-06
PENALTY1	1000	72	3	58	4	1.33	0.85	9.69D-03
POWELLSG	1000	41	3	37	9	3.00	0.35	4.92D-09
POWER	1000	105	10	80	150	7.20	1.18	1.43D-10
QUARTC	1000	188	8	145	21	2.63	1.18	9.10D-09
SINQUAD	1000	171	11	129	25	2.27	4.48	9.92D-07
SROSENBR	1000	16	2	12	3	1.50	0.17	2.45D-16
TOINTGSS	1000	7	1	4	1	1.00	0.22	1.00D+01
TQUARTIC	1000	27	2	19	3	1.50	0.34	4.05D-15
TRIDIA	1000	17	10	10	643	9.80	1.63	7.32D-15
VARDIM	1000	56	3	48	3	1.00	0.41	3.77D-23
VAREIGVL	1000	66	9	57	179	6.89	4.22	7.52D-11
WOODS	1000	75	5	42	14	2.80	0.83	9.56D-15

Key: n = number of variables, $\#f$ = number of function evaluations, $\#min$ = number of minimizations, $\#its$ = total number of iterations, $\#cg$ = total number of CG iterations, $s^{(k)}$ = average subspace dimension, *time* = total CPU time in seconds, f = smallest function value obtained.

Table 11: Results for ISM(p,f,f) on large or hard problems, in which preconditioning is used and the subspace is chosen from the first 10 CG directions.

problem	n	#f	#min	#its	#cg	$s^{(k)}$	time	f
ARWHEAD	1000	23	4	17	5	1.25	0.67	0.00D+00
BDQRTIC	1000	62	7	40	7	1.00	1.55	3.98D+03
BROWNAL	100	125	10	108	134	7.70	3.31	1.05D-11
BRYBND	1000	159	11	97	26	2.36	4.58	7.78D-14
CRAGGLVY	1000	93	9	48	9	1.00	2.76	3.36D+02
DIXMAANA	1500	21	1	18	3	3.00	0.97	1.00D+00
DIXON3DQ	1000	3	1	1	1	1.00	0.05	0.00D+00
DQDRTIC	1000	3	1	1	1	1.00	0.09	1.24D-21
DQRTIC	1000	46	2	40	2	1.00	0.33	5.20D-08
EDENSCH	1000	73	6	37	6	1.00	1.73	6.00D+03
EIGENALS	110	213	13	118	58	4.46	2.24	1.25D-11
ENGVAL1	1000	32	5	21	5	1.00	0.83	1.11D+03
FLETCHCR	1000	12749	1212	10320	1212	1.00	266.31	4.00D-19
FMINSURF	1024	248	9	89	459	9.44	11.15	1.00D+00
FREUROTH	1000	113	6	40	15	2.50	2.55	1.21D+05
GENROSE	1000	6586	598	4955	985	1.65	135.03	1.00D+00
LIARWHD	1000	30	2	24	4	2.00	0.63	1.92D-14
MANCINO	100	111	13	46	14	1.08	97.22	3.03D-18
MOREBV	1000	2	1	1	1	1.00	0.08	7.33D-13
NCB20B	1000	110	15	76	656	6.67	69.90	1.68D+03
NONDIA	1000	116	7	84	17	2.43	2.74	9.90D-01
NONDQUAR	1000	86	11	73	18	1.64	0.98	2.18D-09
PENALTY1	1000	145	7	114	64	5.00	3.76	9.69D-03
POWELLSG	1000	92	14	71	16	1.14	0.94	6.14D-11
POWER	1000	61	4	51	4	1.00	1.37	3.44D-10
QUARTC	1000	46	2	40	2	1.00	0.33	5.20D-08
SINQUAD	1000	184	13	147	41	3.15	5.71	2.93D-10
SROSENBR	1000	36	7	26	7	1.00	0.50	6.14D-25
TOINTGSS	1000	3	1	1	1	1.00	0.14	1.00D+01
TQUARTIC	1000	21	2	18	3	1.50	0.36	2.01D-19
TRIDIA	1000	3	1	1	1	1.00	0.05	1.01D-25
VARDIM	1000	91	14	77	2660	6.86	20.02	5.72D-12
VAREIGVL	1000	70	9	56	154	7.33	6.98	1.25D-09
WOODS	1000	486	35	321	45	1.29	6.27	6.79D-16

Key: n = number of variables, $\#f$ = number of function evaluations, $\#min$ = number of minimizations, $\#its$ = total number of iterations, $\#cg$ = total number of CG iterations, $s^{(k)}$ = average subspace dimension, *time* = total CPU time in seconds, f = smallest function value obtained.

Table 12: Results for ISM(n,e,f) on large or hard problems, in which no preconditioning is used and the subspace is chosen from 10 extreme CG directions.

problem	n	#f	#min	#its	#cg	$s^{(k)}$	time	f
ARWHEAD	1000	11	2	6	3	1.50	0.25	0.00D+00
BDQRTIC	1000	63	10	46	56	5.20	1.89	3.98D+03
BROWNAL	100	5	2	3	3	1.50	0.04	2.45D-08
BRYBND	1000	106	9	81	58	6.33	3.57	1.01D-12
CRAGGLVY	1000	109	12	91	113	7.58	4.42	3.36D+02
DIXMAANA	1500	24	7	15	7	1.00	1.08	1.00D+00
DIXON3DQ	1000	7	4	4	1748	9.00	4.03	1.25D-11
DQDRTIC	1000	10	5	5	12	2.40	0.24	1.91D-14
DQRTIC	1000	188	8	145	21	2.63	1.18	9.10D-09
EDENSCH	1000	65	7	44	25	3.57	1.76	6.00D+03
EIGENALS	110	108	10	70	102	8.40	1.28	1.08D-10
ENGVAL1	1000	52	7	34	22	3.14	1.19	1.11D+03
FLETCHCR	1000	12762	1075	10268	13810	9.93	314.35	8.11D-14
FMINSURF	1024	226	12	107	2140	10.00	20.82	1.00D+00
FREUROTH	1000	223	9	68	45	4.78	4.44	1.21D+05
GENROSE	1000	6642	431	4689	7088	9.90	150.00	1.00D+00
LIARWHD	1000	29	3	24	4	1.33	0.58	1.46D-14
MANCINO	100	68	12	37	17	1.42	67.98	1.34D-18
MOREBV	1000	4	3	3	519	10.00	1.99	1.37D-09
NCB20B	1000	284	18	131	1135	9.78	119.17	1.68D+03
NONDIA	1000	14	3	10	4	1.33	0.38	1.48D-15
NONDQUAR	1000	390	39	303	1636	8.62	7.79	2.67D-06
PENALTY1	1000	72	3	58	4	1.33	0.85	9.69D-03
POWELLSG	1000	41	3	37	9	3.00	0.36	4.92D-09
POWER	1000	105	10	80	150	7.20	1.20	1.43D-10
QUARTC	1000	188	8	145	21	2.63	1.19	9.10D-09
SINQUAD	1000	171	11	129	25	2.27	4.47	9.92D-07
SROSENBR	1000	16	2	12	3	1.50	0.17	2.45D-16
TOINTGSS	1000	7	1	4	1	1.00	0.22	1.00D+01
TQUARTIC	1000	27	2	19	3	1.50	0.35	4.05D-15
TRIDIA	1000	17	10	10	644	9.80	1.67	5.18D-15
VARDIM	1000	56	3	48	3	1.00	0.41	3.77D-23
VAREIGVL	1000	66	9	57	161	6.89	4.09	1.50D-10
WOODS	1000	75	5	42	14	2.80	0.83	9.56D-15

Key: n = number of variables, $\#f$ = number of function evaluations, $\#min$ = number of minimizations, $\#its$ = total number of iterations, $\#cg$ = total number of CG iterations, $s^{(k)}$ = average subspace dimension, *time* = total CPU time in seconds, f = smallest function value obtained.

Table 13: Results for ISM(p,e,f) on large or hard problems, in which preconditioning is used and the subspace is chosen from 10 extreme CG directions.

problem	n	#f	#min	#its	#cg	$s^{(k)}$	time	f
ARWHEAD	1000	23	4	17	5	1.25	0.67	0.00D+00
BDQRTIC	1000	62	7	40	7	1.00	1.57	3.98D+03
BROWNAL	100	122	12	107	140	7.08	3.82	6.64D-13
BRYBND	1000	159	11	97	26	2.36	4.58	7.78D-14
CRAGGLVY	1000	93	9	48	9	1.00	2.73	3.36D+02
DIXMAANA	1500	21	1	18	3	3.00	0.97	1.00D+00
DIXON3DQ	1000	3	1	1	1	1.00	0.05	0.00D+00
DQDRTIC	1000	3	1	1	1	1.00	0.09	1.24D-21
DQRTIC	1000	46	2	40	2	1.00	0.33	5.20D-08
EDENSCH	1000	73	6	37	6	1.00	1.73	6.00D+03
EIGENALS	110	213	13	118	58	4.46	2.24	1.25D-11
ENGVAL1	1000	32	5	21	5	1.00	0.82	1.11D+03
FLETCHCR	1000	12749	1212	10320	1212	1.00	265.25	4.00D-19
FMINSURF	1024	214	11	99	487	9.55	12.12	1.00D+00
FREUROTH	1000	113	6	40	15	2.50	2.55	1.21D+05
GENROSE	1000	6586	598	4955	985	1.65	135.23	1.00D+00
LIARWHD	1000	30	2	24	4	2.00	0.63	1.92D-14
MANCINO	100	111	13	46	14	1.08	97.85	3.03D-18
MOREBV	1000	2	1	1	1	1.00	0.08	7.33D-13
NCB20B	1000	117	15	78	520	6.67	62.12	1.68D+03
NONDIA	1000	116	7	84	17	2.43	2.83	9.90D-01
NONDQUAR	1000	86	11	73	18	1.64	1.00	2.18D-09
PENALTY1	1000	244	16	187	155	5.25	7.70	9.69D-03
POWELLSG	1000	92	14	71	16	1.14	0.95	6.14D-11
POWER	1000	61	4	51	4	1.00	1.38	3.44D-10
QUARTC	1000	46	2	40	2	1.00	0.34	5.20D-08
SINQUAD	1000	184	13	147	41	3.15	5.76	2.93D-10
SROSENBR	1000	36	7	26	7	1.00	0.50	6.14D-25
TOINTGSS	1000	3	1	1	1	1.00	0.14	1.00D+01
TQUARTIC	1000	21	2	18	3	1.50	0.36	2.01D-19
TRIDIA	1000	3	1	1	1	1.00	0.05	1.01D-25
VARDIM	1000	105	19	89	2842	7.21	23.54	4.61D-13
VAREIGVL	1000	84	10	61	269	7.60	8.84	3.95D-10
WOODS	1000	486	35	321	45	1.29	6.27	6.79D-16

Key: n = number of variables, $\#f$ = number of function evaluations, $\#min$ = number of minimizations, $\#its$ = total number of iterations, $\#cg$ = total number of CG iterations, $s^{(k)}$ = average subspace dimension, *time* = total CPU time in seconds, f = smallest function value obtained.

Table 14: Results for ISM(n,f,a) on large or hard problems, in which no preconditioning is used and the subspace dimension is chosen automatically from the first CG directions.

problem	n	#f	#min	#its	#cg	$s^{(k)}$	time	f
ARWHEAD	1000	11	2	6	3	2.00	0.25	0.00D+00
BDQRTIC	1000	52	11	40	62	2.18	1.69	3.98D+03
BROWNAL	100	5	2	3	3	2.00	0.04	2.45D-08
BRYBND	1000	55	10	37	74	2.40	2.27	8.21D-14
CRAGGLVY	1000	64	12	52	122	3.17	2.85	3.36D+02
DIXMAANA	1500	22	7	14	7	2.00	1.00	1.00D+00
DIXON3DQ	1000	7	4	4	1748	23.00	3.98	1.25D-11
DQDRTIC	1000	10	5	5	12	2.40	0.24	1.91D-14
DQRTIC	1000	58	9	45	18	2.00	0.38	1.75D-08
EDENSCH	1000	60	8	33	29	2.50	1.47	6.00D+03
EIGENALS	110	96	15	74	138	3.87	1.37	1.23D-11
ENGVAL1	1000	33	8	27	23	2.25	0.91	1.11D+03
FLETCHCR	1000	7720	1258	6300	14645	2.04	194.48	1.07D-12
FMINSURF	1024	190	11	106	2346	12.09	23.10	1.00D+00
FREUROTH	1000	59	9	32	25	2.44	1.75	1.21D+05
GENROSE	1000	5170	497	3674	7780	4.91	128.63	1.00D+00
LIARWHD	1000	25	5	19	8	2.00	0.54	7.19D-14
MANCINO	100	31	11	20	13	2.00	39.91	1.01D-16
MOREBV	1000	4	3	3	519	11.00	2.13	1.37D-09
NCB20B	1000	147	21	89	1095	2.71	103.56	1.68D+03
NONDIA	1000	13	3	10	4	2.00	0.37	1.48D-15
NONDQUAR	1000	280	47	231	2491	2.28	8.75	9.81D-07
PENALTY1	1000	79	14	58	19	2.00	1.03	9.69D-03
POWELLSG	1000	58	11	49	32	2.27	0.56	5.24D-11
POWER	1000	94	10	72	140	5.20	1.13	6.64D-10
QUARTC	1000	58	9	45	18	2.00	0.39	1.75D-08
SINQUAD	1000	186	34	139	67	2.21	5.91	2.02D-06
SROSENBR	1000	16	5	13	7	2.00	0.21	1.88D-15
TOINTGSS	1000	7	1	4	1	2.00	0.23	1.00D+01
TQUARTIC	1000	44	9	36	13	2.00	0.69	3.27D-16
TRIDIA	1000	17	10	10	591	9.70	1.51	7.14D-14
VARDIM	1000	58	9	40	9	2.00	0.40	1.14D-19
VAREIGVL	1000	52	9	44	162	3.78	3.66	1.77D-10
WOODS	1000	37	7	29	17	2.57	0.60	2.76D-11

Key: n = number of variables, $\#f$ = number of function evaluations, $\#min$ = number of minimizations, $\#its$ = total number of iterations, $\#cg$ = total number of CG iterations, $s^{(k)}$ = average subspace dimension, *time* = total CPU time in seconds, f = smallest function value obtained.

Table 15: Results for ISM(p,f,a) on large or hard problems, in which preconditioning is used and the subspace dimension is chosen automatically from the first CG directions.

problem	n	#f	#min	#its	#cg	$s^{(k)}$	time	f
ARWHEAD	1000	22	4	15	5	2.00	0.63	7.22D-14
BDQRTIC	1000	38	7	29	8	2.00	1.27	3.98D+03
BROWNAL	100	55	10	46	126	2.80	3.08	1.75D-11
BRYBND	1000	61	10	36	25	2.40	2.33	6.05D-14
CRAGGLVY	1000	54	10	42	10	2.00	2.06	3.36D+02
DIXMAANA	1500	20	6	15	8	2.17	1.23	1.00D+00
DIXON3DQ	1000	3	1	1	1	2.00	0.05	0.00D+00
DQDRTIC	1000	3	1	1	1	2.00	0.09	1.24D-21
DQRTIC	1000	48	8	40	8	2.00	0.41	8.23D-09
EDENSCH	1000	33	6	26	6	2.00	1.09	6.00D+03
EIGENALS	110	122	16	79	55	2.56	1.62	4.21D-11
ENGVAL1	1000	26	5	17	5	2.00	0.70	1.11D+03
FLETCHCR	1000	7053	1246	6220	1246	2.00	161.54	3.32D-16
FMINSURF	1024	199	11	77	540	5.91	11.94	1.00D+00
FREUROTH	1000	65	7	29	8	2.00	1.68	1.21D+05
GENROSE	1000	3682	593	2950	957	2.00	81.75	1.00D+00
LIARWHD	1000	124	15	65	31	2.07	2.26	1.11D+01
MANCINO	100	53	14	32	16	2.00	62.23	1.09D-18
MOREBV	1000	2	1	1	1	2.00	0.08	7.33D-13
NCB20B	1000	94	15	69	534	5.20	59.05	1.68D+03
NONDIA	1000	142	19	99	44	2.53	3.69	9.90D-01
NONDQUAR	1000	124	18	102	57	2.39	1.53	1.07D-07
PENALTY1	1000	77	13	60	60	2.08	4.30	9.69D-03
POWELLSG	1000	73	14	63	16	2.07	0.81	1.17D-09
POWER	1000	48	8	40	8	2.00	2.22	1.05D-11
QUARTC	1000	48	8	40	8	2.00	0.41	8.23D-09
SINQUAD	1000	283	56	215	177	2.29	11.50	6.90D-08
SROSENBR	1000	30	7	23	7	2.00	0.44	9.58D-26
TOINTGSS	1000	3	1	1	1	2.00	0.14	1.00D+01
TQUARTIC	1000	26	5	20	8	2.00	0.49	3.50D-16
TRIDIA	1000	3	1	1	1	2.00	0.05	1.01D-25
VARDIM	1000	78	22	57	2887	2.27	24.91	5.87D-13
VAREIGVL	1000	47	9	40	147	3.67	6.06	1.57D-10
WOODS	1000	143	23	105	25	2.00	2.19	1.21D-26

Key: n = number of variables, $\#f$ = number of function evaluations, $\#min$ = number of minimizations, $\#its$ = total number of iterations, $\#cg$ = total number of CG iterations, $s^{(k)}$ = average subspace dimension, *time* = total CPU time in seconds, f = smallest function value obtained.

Table 16: Results for ISM(n,e,a) on large or hard problems, in which no preconditioning is used and the subspace dimension is chosen automatically from extreme CG directions.

problem	n	#f	#min	#its	#cg	$s^{(k)}$	time	f
ARWHEAD	1000	11	2	6	3	2.00	0.25	0.00D+00
BDQRTIC	1000	52	11	40	62	2.18	1.69	3.98D+03
BROWNAL	100	5	2	3	3	2.00	0.04	2.45D-08
BRYBND	1000	53	10	36	72	2.40	2.21	9.91D-14
CRAGGLVY	1000	66	12	55	126	3.58	3.00	3.36D+02
DIXMAANA	1500	22	7	14	7	2.00	1.00	1.00D+00
DIXON3DQ	1000	7	4	4	1748	23.00	4.19	1.25D-11
DQDRTIC	1000	10	5	5	12	2.40	0.24	1.91D-14
DQRTIC	1000	58	9	45	18	2.00	0.38	1.75D-08
EDENSCH	1000	60	8	33	29	2.50	1.47	6.00D+03
EIGENALS	110	94	17	73	127	3.47	1.35	5.34D-11
ENGVAL1	1000	33	8	27	23	2.25	0.93	1.11D+03
FLETCHCR	1000	7722	1262	6300	14691	2.03	195.36	3.98D-15
FMINSURF	1024	216	12	100	2259	8.33	22.14	1.00D+00
FREUROTH	1000	80	9	32	28	2.56	1.92	1.21D+05
GENROSE	1000	5445	475	3895	7394	5.71	126.73	1.00D+00
LIARWHD	1000	25	5	19	8	2.00	0.49	7.19D-14
MANCINO	100	31	11	20	13	2.00	38.06	1.01D-16
MOREBV	1000	4	3	3	519	11.00	1.97	1.37D-09
NCB20B	1000	159	21	95	1139	3.62	105.14	1.68D+03
NONDIA	1000	13	3	10	4	2.00	0.35	1.48D-15
NONDQUAR	1000	341	53	257	3507	2.42	11.35	1.64D-06
PENALTY1	1000	79	14	58	19	2.00	0.96	9.69D-03
POWELLSG	1000	58	11	49	32	2.27	0.52	5.24D-11
POWER	1000	94	10	72	140	5.20	1.06	6.64D-10
QUARTC	1000	58	9	45	18	2.00	0.39	1.75D-08
SINQUAD	1000	208	41	156	88	2.20	6.43	1.71D-07
SROSENBR	1000	16	5	13	7	2.00	0.19	1.88D-15
TOINTGSS	1000	7	1	4	1	2.00	0.21	1.00D+01
TQUARTIC	1000	44	9	36	13	2.00	0.65	3.27D-16
TRIDIA	1000	17	10	10	591	9.40	1.57	8.20D-14
VARDIM	1000	58	9	40	9	2.00	0.38	1.14D-19
VAREIGVL	1000	54	10	46	228	4.40	4.18	2.07D-11
WOODS	1000	37	7	29	17	2.57	0.54	2.76D-11

Key: n = number of variables, $\#f$ = number of function evaluations, $\#min$ = number of minimizations, $\#its$ = total number of iterations, $\#cg$ = total number of CG iterations, $s^{(k)}$ = average subspace dimension, *time* = total CPU time in seconds, f = smallest function value obtained.

Table 17: Results for ISM(p,e,a) on large or hard problems, in which preconditioning is used and the subspace dimension is chosen automatically from extreme CG directions.

problem	n	#f	#min	#its	#cg	$s^{(k)}$	time	f
ARWHEAD	1000	22	4	15	5	2.00	0.59	7.22D-14
BDQRTIC	1000	38	7	29	8	2.00	1.17	3.98D+03
BROWNAL	100	56	11	48	140	3.27	3.28	3.25D-13
BRYBND	1000	66	11	40	26	2.27	2.58	2.06D-14
CRAGGLVY	1000	54	10	42	10	2.00	2.04	3.36D+02
DIXMAANA	1500	20	6	15	8	2.17	1.20	1.00D+00
DIXON3DQ	1000	3	1	1	1	2.00	0.05	0.00D+00
DQDRTIC	1000	3	1	1	1	2.00	0.09	1.24D-21
DQRTIC	1000	48	8	40	8	2.00	0.42	8.23D-09
EDENSCH	1000	33	6	26	6	2.00	1.08	6.00D+03
EIGENALS	110	112	16	70	56	2.44	1.53	2.50D-13
ENGVAL1	1000	26	5	17	5	2.00	0.70	1.11D+03
FLETCHCR	1000	7053	1246	6220	1246	2.00	162.80	3.32D-16
FMINSURF	1024	190	13	86	549	6.54	12.77	1.00D+00
FREUROTH	1000	65	7	29	8	2.00	1.67	1.21D+05
GENROSE	1000	3682	593	2950	957	2.00	81.89	1.00D+00
LIARWHD	1000	124	15	65	31	2.07	2.24	1.11D+01
MANCINO	100	53	14	32	16	2.00	64.59	1.09D-18
MOREBV	1000	2	1	1	1	2.00	0.08	7.33D-13
NCB20B	1000	101	15	70	505	4.00	57.74	1.68D+03
NONDIA	1000	142	19	99	44	2.53	3.66	9.90D-01
NONDQUAR	1000	122	17	96	46	2.47	1.46	9.13D-08
PENALTY1	1000	80	14	64	62	2.07	4.60	9.69D-03
POWELLSG	1000	73	14	63	16	2.07	0.80	1.17D-09
POWER	1000	48	8	40	8	2.00	2.24	1.05D-11
QUARTC	1000	48	8	40	8	2.00	0.41	8.23D-09
SINQUAD	1000	440	93	301	281	2.15	17.97	1.11D-08
SROSENBR	1000	30	7	23	7	2.00	0.44	9.58D-26
TOINTGSS	1000	3	1	1	1	2.00	0.14	1.00D+01
TQUARTIC	1000	26	5	20	8	2.00	0.49	3.50D-16
TRIDIA	1000	3	1	1	1	2.00	0.05	1.01D-25
VARDIM	1000	82	22	59	2804	2.32	24.92	1.24D-12
VAREIGVL	1000	49	9	42	134	3.67	6.03	3.26D-10
WOODS	1000	143	23	105	25	2.00	2.20	1.21D-26

Key: n = number of variables, $\#f$ = number of function evaluations, $\#min$ = number of minimizations, $\#its$ = total number of iterations, $\#cg$ = total number of CG iterations, $s^{(k)}$ = average subsparce dimension, *time* = total CPU time in seconds, f = smallest function value obtained.