Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization

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

March 31, 1993

Abstract. In this paper, we describe the algorithmic options of Release A of LANCELOT, a new Fortran package for large-scale nonlinear optimization. We then present the results of intensive numerical tests and discuss the relative merits of the options. The experiments described involve both academic and applied problems. Conclusions are finally proposed, both specific to LANCELOT and of more general scope.

[1] IBM T.J. Watson Research Centre
Yorktown Heights, USA

[2] Rutherford Appleton Laboratory Oxfordshire, England

[3] Department of Mathematics,
Facultés Universitaires ND de la Paix, Namur, Belgium

Keywords: Large-scale problems, nonlinear optimization, numerical algorithms.

[0] This research 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.

Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization

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

Report 92/16 March 31, 1993

Abstract. In this paper, we describe the algorithmic options of Release A of LANCELOT, a new Fortran package for large-scale nonlinear optimization. We then present the results of intensive numerical tests and discuss the relative merits of the options. The experiments described involve both academic and applied problems. Conclusions are finally proposed, both specific to LANCELOT and of more general scope.

[1] IBM T.J. Watson Research Centre
Yorktown Heights, USA

[2] Rutherford Appleton Laboratory Oxfordshire, England

[3] Department of Mathematics,
Facultés Universitaires ND de la Paix, Namur, Belgium

Keywords: Large-scale problems, nonlinear optimization, numerical algorithms.

[0] This research 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.

1 Introduction

Research in large-scale optimization has been, in recent years, a major subject of interest within the mathematical programming community, as is clear from the programs of the main conferences and symposia on optimization techniques during this period. One such project, LANCELOT, was initiated by the authors of this paper [12] and has resulted in both theoretical contributions and software for large nonlinear optimization problems. A detailed description of the algorithms developed and implemented in LANCELOT, the resulting new Fortran package, is presented in [17]. The purpose of the present paper is to report on the numerical experiments obtained with this software on a sizeable collection of test problems, and to draw some first conclusions on the respective merits of the algorithmic options available in the package. A comparison of LANCELOT and MINOS [48] will be reported on separately [4].

The paper is organized as follows. Section 2 briefly presents the main features and structure of LANCELOT. Section 3 contains a general description of SBMIN, the kernel algorithm for the software that handles simple bounds. AUGLG, the component that handles the extension to general constraints, is then presented in Section 4. Section 5 discusses the various algorithmic options that are available within the package. Section 6 presents the testing framework and the strategy used to analyze the results. These results are then discussed in more detail in Section 7, where the efficiency and robustness of various algorithmic options is compared. Finally, some conclusions and perspectives are drawn in Section 8.

2 General features and structure of the LANCELOT package

2.1 Package presentation

The purpose of the LANCELOT package is to solve the general nonlinear programming problem

$$\min_{x \in \Re^n} f(x) \tag{2.1}$$

subject to the constraints

$$c(x) = 0, \tag{2.2}$$

and to the simple bounds

$$l_i \leq x_i \leq u_i, \quad (i = 1, \ldots, n), \tag{2.3}$$

where f, and c are assumed to be smooth functions from $\Re^n$ into $\Re$ and from $\Re^n$ into $\Re^m$ respectively. *The package is specially intended for problems where n and/or m are large.* Indeed it exploits the *(group) partially separable* structure (see [12]) of most large-scale optimization problems. The algorithms are designed to provide convergence of the generated iterates to local minimizers from all starting points.

There is no loss in assuming that all the general constraints are equations, as inequality constraints may easily be transformed to equations by the addition of extra slack or surplus variables (see, for example, [31, Section 5.6]). Indeed, LANCELOT automatically transforms inequality constraints to equations. This technique is extensively used in simplex-like methods for large-scale linear and nonlinear programs.

General features include facilities to compute numerical derivatives, an analytical derivative checker and an automated restart. The software also uses a full reverse communication interface for greater flexibility and adaptability.

The package is written in standard ANSI Fortran77. It has already been ported to CRAY supercomputers, Digital VAX minis and RISC workstations, IBM VM/CMS and RISC6000 and SUN workstations. A fully automated installation procedure is supported for all these machines/systems. Single and double precision versions are available. The program's dimensions are also adaptable to fit within machines with different memory sizes.

Full information on the package is available in [17]. Interested parties should contact one of the authors.

2.2 The algorithmic structure of the package

Because the purpose of the paper is to discuss the relative merits of several algorithmic options within the package, it is necessary to provide first a general description of the numerical methods used. The structure of the LANCELOT algorithms is summarized in Figure 1.

The package (whose algorithmic components appear in the rounded box) reads the problem as a set of data and Fortran subroutines (for computing function and derivatives values, as well as other problem related tasks). The way in which these subroutines and the associated datafile are produced is not the subject of this paper. It suffices to say that they can be written directly by the user, or obtained as the result of the automated interpretation of the problem expressed in a more friendly Standard Input Format. These techniques are described in detail in [17] and will not be discussed further.

We will rather concentrate on the algorithms used by LANCELOT to solve the problem, once properly specified. As suggested by the picture, LANCELOT either uses an augmented Lagrangian approach if constraints of the type (2.2) are present, or directly attempts to solve problems whose only constraints are simple bounds, (2.3).

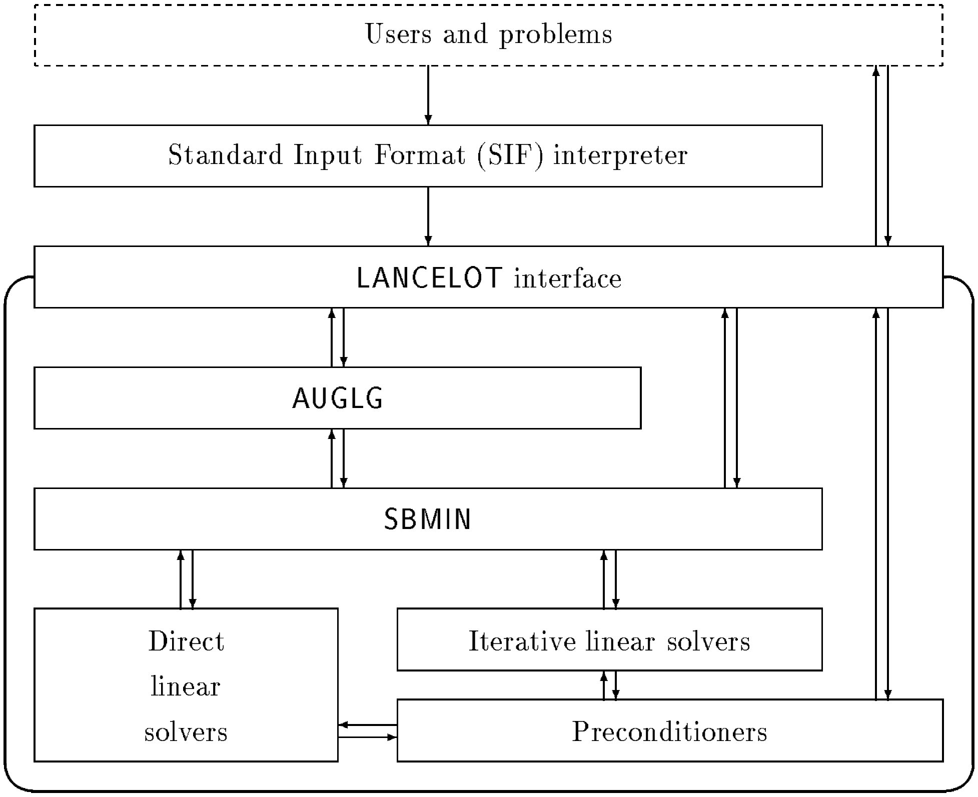

Figure 1: Structure of the LANCELOT package

The augmented Lagrangian algorithm AUGLG is described in Section 4. Its convergence theory has been analyzed in [13] and [18]. This theory guarantees that, under standard assumptions, the sequence of iterates calculated by the algorithm converges to a local minimizer of the problem. This augmented Lagrangian method proceeds by solving a sequence of suitably defined nonlinear optimization problems with simple bound constraints. We will call these iterations of the augmented Lagrangian algorithm *major iterations*.

If the considered problem only possesses simple bound constraints , then a specialized algorithm, SBMIN, can be applied. This algorithm is of trust region type and is presented in Section 3. Its strong convergence properties have been analyzed in [10] and [57]. At the heart of SBMIN, quadratic problems with bound constraints (BQP) are solved repeatedly. In fact, one such BQP is approximately solved at every SBMIN iteration. We call these *minor iterations*.

The process of (approximately) solving the BQP involves the (approximate) solution of a linear system of equations. This can be achieved by applying either direct or iterative linear solvers. The latter typically requires preconditioning, which in turn might call specialized versions of the direct solvers, as is shown in the figure above. The iterative technique used with the package is preconditioned conjugate gradients. Iterations at this level are simply called *cg-iterations*. Note that some form of preconditioning might require a very problem specific technique; hence the possibility to return to the user level for such a calculation.

The three nested iteration levels (major iterations at the augmented Lagrangian level, minor iterations at the SBMIN level, and cg-iterations at the BQP level) are illustrated in Figure 2, where the dashed boxes indicate iteration levels that need not be present for all problems and all choices of algorithmic options.

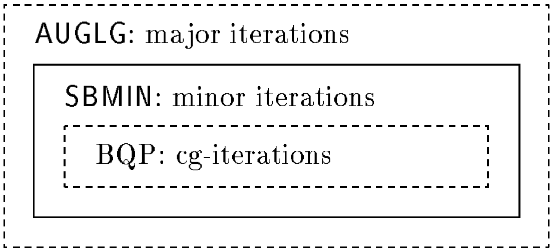

Figure 2: The nested iterations levels within LANCELOT

Because the bulk of the computational work is in the two inner iterations, we will concentrate on these levels in what follows.

3 A general description of SBMIN

SBMIN is a method for solving the bound-constrained minimization problem

$$\underset{x \in \Re^n}{\text{minimize}} \quad f(x) \tag{3.1}$$

subject to the simple bound constraints

$$l_i \leq x_i \leq u_i, \quad 1 \leq i \leq n. \tag{3.2}$$

Here, f is assumed to be twice-continuously differentiable and any of the bounds in (3.2) may be infinite. We will denote the vector of first partial derivatives, $\nabla_x f(x)$, by $g(x)$ and the Hessian matrix, $\nabla_{xx} f(x)$, will be denoted by $H(x)$. We shall refer to the set of points which satisfy (3.2) as the *feasible box* and any point lying in the feasible box is said to be *feasible*.

SBMIN is an iterative method. At the end of the k-th iteration, an estimate of the solution, $x^{(k)}$, satisfying the simple bounds (3.2), is given. The purpose of the $(k+1)$-st iteration is to find a feasible iterate $x^{(k+1)}$ which is a significant improvement on $x^{(k)}$.

In the $(k+1)$-st iteration, we build a quadratic model of our (possibly) nonlinear objective function, $f(x)$. This model takes the form

$$m^{(k)}(x) = f(x^{(k)}) + g(x^{(k)})^T(x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T B^{(k)}(x - x^{(k)}), \tag{3.3}$$

where $B^{(k)}$ is a symmetric approximation to the Hessian matrix $H(x^{(k)})$. We also define a scalar $\Delta^{(k)}$, the *trust-region radius*, which defines the *trust region*,

$$\|x - x^{(k)}\| \leq \Delta^{(k)}, \tag{3.4}$$

within which we trust that the values of $m^{(k)}(x)$ and $f(x)$ will generally agree sufficiently. An appropriate range of values for the trust-region radius is accumulated as the minimization proceeds.

The $(k+1)$-st iteration proceeds in a number of stages. These may be summarized, in order, as:

1. Test for convergence (see Section 3.1).
2. Find an approximation to the generalized Cauchy point of the quadratic model, within the intersection of the feasible box and the trust region (see Section 3.2).
3. Obtain a new point which further reduces the quadratic model within the intersection of the feasible box and the trust region (see Section 3.3).
4. Test whether there is a general agreement between the values of the model and true objective function at the new point. If so, accept the new point as the next iterate. Otherwise, retain the existing iterate as the next iterate. In either case, adjust the trust region radius as appropriate (see Section 3.4).

Clearly, this is a very sketchy description — indeed, some of the terminology has not even been defined — and we need to elaborate further.

3.1 The test for convergence

The first-order necessary conditions for a feasible point x^* to solve the problem (3.1)–(3.2) require that the projected gradient at x^* be zero. The *projected gradient* of $f(x)$ into the feasible box (3.2) is defined to be

$$x - P(x - g(x), l, u), \tag{3.5}$$

where the projection operator, $P(x, l, u)$ is defined componentwise by

$$P(x, l, u)_i = \begin{cases} l_i & \text{if } x_i < l_i \\ u_i & \text{if } x_i > u_i \\ x_i & \text{otherwise.} \end{cases} \tag{3.6}$$

Notice that, in the unconstrained case where all the bounds (3.2) are infinite, this is merely the well-known condition that the gradient must vanish at a minimizer.

One may then gauge the convergence of the method by the size of the projected gradient at $x^{(k)}$. For instance, one might stop if the condition

$$\|x^{(k)} - P(x^{(k)} - g(x^{(k)}), l, u)\|_\infty \leq \epsilon_g, \tag{3.7}$$

holds for some appropriate small convergence tolerance ϵ_g. Alternatively, following [22, p. 160], one might consider the relative projected gradient with i-th component

$$r_i(x) = \frac{(x_i - P(x - g(x), l, u)_i) \max(x_i, x_i^{\text{typical}})}{\max(f(x), f^{\text{typical}})} \tag{3.8}$$

for "typical" values x^{typical} and f^{typical} (which may, of course, be sometimes difficult to choose). In this case, it might be appropriate to stop if the condition

$$\|r(x^{(k)})\|_\infty \leq \epsilon_r, \tag{3.9}$$

is satisfied for some appropriate small convergence tolerance ϵ_r.

In LANCELOT, the relative convergence test (3.9) is used when suitable scalings x^{typical} and f^{typical} are available (see Section 5.1). In all other cases, the test (3.7) is used.

3.2 The generalized Cauchy point

The approximate minimization of the quadratic model (3.3) within the intersection of the feasible box and the trust region at the $(k+1)$-st iteration is accomplished in two stages. In the first, we obtain an approximation to the so called generalized *Cauchy point.* This point is important for two reasons. Firstly, convergence of the algorithm to a point at which the projected gradient is zero can be guaranteed provided the value of the quadratic model at the end of the iteration is no larger than that at the generalized Cauchy point (see [10]). Secondly, the set of variables which lie on their bounds at the generalized Cauchy point as the algorithm proceeds often provide an excellent prediction of those which will ultimately be fixed at the solution to the problem. It is thus important that an adequate approximation to such a point be identified.

Let $D^{(k)}$ be a positive definite diagonal scaling matrix and let

$$\bar{g}^{(k)} = D^{(k)} g(x^{(k)}). \tag{3.10}$$

Now define the *Cauchy arc*, $x^{(k)}(t)$, by

$$x^{(k)}(t) = P(x^{(k)} - t\bar{g}^{(k)}, l, u) \tag{3.11}$$

for all values of the parameter $t \geq 0$. Considering the arc as t increases from zero, the generalized Cauchy point is defined to be $x(t_e)$, where t_e is the first local minimizer of $m^{(k)}(x^{(k)}(t))$, the quadratic model along the arc, subject to the trust region constraint (3.4) being satisfied at $x^{(k)}(t)$. An efficient algorithm for this calculation, when $\|.\|$ is the infinity-norm (the LANCELOT default), is given in [11].

It is not necessary that the generalized Cauchy point be calculated exactly. Indeed, a number of authors have considered approximations which are sufficient to guarantee convergence (see [6], [7], [8], [44], [57]). We also provide, as an option, the possibility to use the approximation suggested by Moré in [44]. Let $\gamma > 0$, $0 < \beta < 1$ and $0 < \sigma < 1$. Then we choose the approximation $x(t_i)$, where t_i is of the form $\gamma\beta^{m_s}$ and where m_s is the smallest nonnegative integer for which

$$m^{(k)}(x^{(k)}(t)) \leq m^{(k)}(x^{(k)}) + \sigma g(x^{(k)})^T (x^{(k)}(t) - x^{(k)}) \tag{3.12}$$

and

$$\|x^{(k)}(t) - x^{(k)}\| \leq \beta\Delta^{(k)}. \tag{3.13}$$

We shall refer to such a point as an *approximate* generalized Cauchy point. Notice that the generalized Cauchy point may not satisfy equation (3.12). Thus there is a possibility of different behaviour for algorithms which use the true and approximate generalized Cauchy points.

3.3 Beyond the generalized Cauchy point

We have ensured that SBMIN will converge by determining a suitable approximation to the generalized Cauchy point. We are now concerned that the algorithm should converge at a reasonable rate. This is achieved by, if necessary, further reducing the quadratic model.

Those variables which lie on their bounds at the approximation to the generalized Cauchy point are fixed. Attempts are then made to reduce the quadratic model by changing the values of the remaining free variables. Let $x^{(k,1)}$ be the obtained approximation to the generalized Cauchy point and let $x^{(k,j)}, j = 2, 3, ...$ be distinct points such that:

- $x^{(k,j)}$ lies within the intersection of the feasible box and the trust region;
- those variables which lie on a bound at $x^{(k,1)}$ lie on the same bound at $x^{(k,j)}$;
- $x^{(k,j+1)}$ is constructed from $x^{(k,j)}$ by
  1. determining a nonzero search direction $p^{(k,j)}$ for which
  $$\nabla_x m^{(k)}(x^{(k,j)})^T p^{(k,j)} < 0; \tag{3.14}$$
  2. finding a steplength $\alpha^{(k,j)} > 0$ which minimizes $m^{(k)}(x^{(k,j)} + \alpha p^{(k,j)})$ within the intersection of the feasible box and the trust region; and
  3. setting
  $$x^{(k,j+1)} = x^{(k,j)} + \alpha^{(k,j)} p^{(k,j)}. \tag{3.15}$$

Notice that the one dimensional minimizer of $m^{(k)}(x^{(k,j)} + \alpha p^{(k,j)})$ within the intersection of the feasible box and the trust region is easy to compute. Let the gradient of the model at $x^{(k,j)}$ be

$$g^{(k,j)} \equiv g(x^{(k)}) + B^{(k)}(x^{(k,j)} - x^{(k)}). \tag{3.16}$$

Then the unconstrained line minimizer of the quadratic model is given by

$$\alpha_0^{(k,j)} = \begin{cases} -g^{(k,j)T} p^{(k,j)} / p^{(k,j)T} B^{(k)} p^{(k,j)} & \text{if } p^{(k,j)T} B^{(k)} p^{(k,j)} > 0, \\ \infty & \text{otherwise.} \end{cases} \tag{3.17}$$

Furthermore, the maximum feasible step allowed by the bound (3.2) on the i-th variable is

$$\alpha_i^{(k,j)} = \begin{cases} (l_i - x_i^{(k,j)})/p_i^{(k,j)} & \text{if } p_i^{(k,j)} < 0, \\ (u_i - x_i^{(k,j)})/p_i^{(k,j)} & \text{if } p_i^{(k,j)} > 0, \\ \infty & \text{otherwise,} \end{cases} \tag{3.18}$$

for $1 \leq i \leq n$. Finally, the trust region imposes a bound on the step, $\alpha_{n+1}^{(k,j)} > 0$, where

$$\|x^{(k,j)} + \alpha_{n+1}^{(k,j)} p^{(k,j)}\| = \Delta^{(k)}. \tag{3.19}$$

Therefore the required steplength is

$$\alpha^{(k,j)} = \min_{0 \leq i \leq n+1} \alpha_i^{(k,j)}. \tag{3.20}$$

This process is stopped when the norm of the free gradient of the model at $x^{(k,j)}$ is sufficiently small. The *free gradient of the model* is

$$Q(\nabla_x m^{(k)}(x^{(k,j)}), x^{(k,j)}, l, u), \tag{3.21}$$

where the operator

$$Q(y, x, l, u)_i = \begin{cases} y_i & \text{if } l_i < x_i < u_i, \\ 0 & \text{otherwise.} \end{cases} \tag{3.22}$$

Zeros components of the gradient correspond to variables which lie on their bounds. In LANCELOT, we stop when

$$\|Q(\nabla_x m^{(k)}(x^{(k,j)}), x^{(k,j)}, l, u)\| \leq \|Q(\nabla_x m^{(k)}(x^{(k)}), x^{(k,1)}, l, u)\|^{1.5}, \tag{3.23}$$

which is known (see [41]) to guarantee that the convergence rate of the method is asymptotically superlinear. Notice that the free gradient at the final point is measured against the components of the gradient of the model at the original point which were free at the approximation to the generalized Cauchy point.

There is much flexibility in obtaining a search direction which satisfies (3.14). We determine such a direction by finding an approximation to the minimizer of the quadratic subproblem (3.3), where certain of the variables are fixed on their bounds but the constraints on the remaining variables are ignored. Specifically, let $\mathcal{I}^{(k,j)}$ be a set of indices of the variables which are to be fixed, let e_i be the i-th column of the n by n identity matrix I and let $\bar{I}^{(k,j)}$ be the matrix made up of columns e_i, $i \notin \mathcal{I}^{(k,j)}$. Now define

$$\bar{g}^{(k,j)} \equiv \bar{I}^{(k,j)T} g^{(k,j)} \text{ and } \bar{B}^{(k,j)} \equiv \bar{I}^{(k,j)T} B^{(k,j)} \bar{I}^{(k,j)}. \tag{3.24}$$

Then the quadratic model (3.3) at $x^{(k,j)} + p$, considered as a function of the free variables $\bar{p} \equiv \bar{I}^{(k,j)T} p$ is,

$$\bar{m}^{(k,j)}(\bar{p}) = m^{(k)}(x^{(k,j)}) + \bar{g}^{(k,j)T} \bar{p} + \frac{1}{2} \bar{p}^T \bar{B}^{(k,j)} \bar{p}. \tag{3.25}$$

We may attempt to minimize (3.25) using either a direct or iterative method.

In a direct minimization of (3.25), one factorizes the coefficient matrix $\bar{B}^{(k,j)}$. If the factors indicate that the matrix is positive definite, the Newton equations

$$\bar{B}^{(k,j)} \bar{p}^{(k,j)} = -\bar{g}^{(k,j)} \tag{3.26}$$

may be solved and the required search direction $p^{(k,j)} = \bar{I}^{(k,j)}\bar{p}^{(k,j)}$ recovered. If, on the other hand, the matrix is merely positive semi-definite, a direction of linear infinite descent (dolid) or a weak solution to the Newton equations can be determined. Finally, if the matrix is truly indefinite, a direction of negative curvature (donc) may be obtained.

In an iterative minimization of (3.25), the index set $\mathcal{I}^{(k,j)}$ may stay constant over a number of iterations, while at each iteration the search direction may be calculated from the current model gradient and Hessian $\bar{B}^{(k,j)}$ and previous search directions. The iterative method used in LANCELOT is the method of conjugate gradients. The convergence of such a method may be accelerated by preconditioning (see below) . In fact the boundary between a good preconditioned iterative method and a direct method is quite blurred.

3.4 Accepting the new point and other bookkeeping

A point $x^{(k,j)}$, which gives a sufficient reduction in the quadratic model, has thus been found. Of course, we are really interested in reducing the true objective function $f(x)$, not the model. The success or failure of the iteration may be measured by computing the ratio of the actual reduction in the objective function to that predicted by the model

$$\rho^{(k)} = \frac{f(x^{(k)}) - f(x^{(k,j)})}{m^{(k)}(x^{(k)}) - m^{(k)}(x^{(k,j)})}. \tag{3.27}$$

If this ratio is negative or small relative to one, the iteration is viewed as a failure. We call such an iteration *unsuccessful.* Conversely if the ratio is large, the model predicted a reasonable decrease in the objective function and the iteration has been *successful* (even though, if the ratio is significantly larger than one, the model was not an accurate approximation to f).

Let $0 < \mu < 1$. We choose to update $x^{(k+1)}$ as follows:

$$x^{(k+1)} = \begin{cases} x^{(k,j)} & \text{if } \rho^{(k)} > \mu, \\ x^{(k)} & \text{otherwise.} \end{cases} \tag{3.28}$$

We note that, as $\mu > 0$, this may prevent the algorithm from accepting the lowest point encountered so far as the new iterate. One could circumvent this drawback by explicitly keeping the overall lowest point found and returning it as the required solution. However, this requires additional storage and has not proved beneficial in our experience.

We also need to update the trust-region radius. Again, if ρ is small or negative, the model has not predicted the behaviour of the true objective function well, while if the ratio is close to one, a good prediction has been obtained. In the former case, the region in which the quadratic approximation is trusted must be reduced and the radius consequently decreased. In the latter case, their may be reason to believe that the region in which we may trust our model is larger than we previously thought and the radius is increased as a consequence.

Again let $\mu < \eta < 1$ and $0 < \gamma_0 < 1 \leq \gamma_2$. We choose to update the trust region radius according to the formula

$$\Delta^{(k+1)} = \begin{cases} \gamma_0^{(k)}\Delta^{(k)} & \text{if } \rho^{(k)} \leq \mu, \\ \Delta^{(k)} & \text{if } \mu < \rho^{(k)} < \eta, \\ \gamma_2^{(k)}\Delta^{(k)} & \text{otherwise,} \end{cases} \tag{3.29}$$

where $\gamma_0^{(k)} \in [\gamma_0, 1)$ and $\gamma_2^{(k)} \in [1, \gamma_2]$.

If the iterate has changed, we need to recompute the gradient at the new point. In this case we also form a new second derivative approximation.

4 A general description of AUGLG

AUGLG is a method for solving the generally-constrained minimization problem,

$$\underset{x \in \Re^n}{\text{minimize}} \quad f(x) \tag{4.1}$$

subject to the general (possibly nonlinear) constraints

$$c_j(x) = 0, \quad 1 \leq j \leq m, \tag{4.2}$$

and the simple bounds

$$l_i \leq x_i \leq u_i, \quad 1 \leq i \leq n. \tag{4.3}$$

Here, f and the c_j are all assumed to be twice-continuously differentiable and any of the bounds in (4.3) may be infinite.

AUGLG makes repeated use of SBMIN. The objective function and general constraints are combined into a composite function, the *augmented Lagrangian function*,

$$\Phi(x, \lambda, S, \mu) = f(x) + \sum_{i=1}^{m} \lambda_i c_i(x) + \frac{1}{2\mu} \sum_{i=1}^{m} s_{ii} c_i(x)^2, \tag{4.4}$$

where the components λ_i of the vector λ are known as *Lagrange multiplier estimates*, the entries s_{ii} of the diagonal matrix S are positive scaling factors, and μ is known as the *penalty parameter.*

The constrained minimization problem (4.1)–(4.3) is now solved by finding approximate minimizers of Φ subject to the simple bounds (4.3), for a carefully constructed sequence of Lagrange multiplier estimates, constraint scaling factors and penalty parameters.

The $k+1$-st major iteration of AUGLG is made up of three steps. At the start of the iteration, Lagrange multiplier estimates, $\lambda^{(k)}$, constraint scaling factors, $S^{(k)}$, and a penalty parameter $\mu^{(k)}$ are given. The steps performed may be summarized, in order, as follows:

1. Test for convergence (see Section 4.1).

2. Use SBMIN to find an approximate minimizer, $x^{(k+1)}$, of the augmented Lagrangian function $\Phi(x, \lambda^{(k)}, S^{(k)}, \mu^{(k)})$ in the feasible box, (4.3) (see Section 4.2).

3. Update the Lagrange multiplier estimates, constraint scaling factors, penalty parameter and convergence tolerances (see Section 4.3).

We now consider these steps in more detail.

4.1 Convergence of the augmented Lagrangian method

The first-order necessary conditions for a feasible point x^* to solve the problem (4.1)–(4.3) require that there are *Lagrange multipliers*, λ^*, for which the projected gradient of the Lagrangian function at x^* and λ^*, and the general constraints (4.2) at x^*, vanish. The *Lagrangian function* is the function

$$L(x,\lambda) = f(x) + \sum_{i=1}^{m} \lambda_i c_i(x). \qquad (4.5)$$

The projected gradient is given by (3.5) but now with the gradient of the Lagrangian function, $\nabla_x L(x,\lambda)$, replacing the gradient of f.

One may then assess the convergence of the augmented Lagrangian method by the size of the projected gradient and constraints at $x^{(k)}$ and $\lambda^{(k)}$. For instance, one might stop if the conditions

$$\|x^{(k)} - P(x^{(k)} - \nabla_x L(x^{(k)}, \lambda^{(k)}), l, u)\|_\infty \leq \epsilon_l, \qquad (4.6)$$

and

$$\|c(x^{(k)})\|_\infty \leq \epsilon_c, \qquad (4.7)$$

hold for some appropriate small convergence tolerances ϵ_l and ϵ_c.

Alternatively, one might consider the values of the relative constraint functions, $c^{\text{relative}}(x)$, where

$$c_i^{\text{relative}}(x) = c_i(x)/c_i^{\text{typical}} \qquad (4.8)$$

for "typical" values c^{typical}, and the relative projected gradient of the Lagrangian and stop when both of these quantities are small in norm.

In LANCELOT, the relative convergence tests ((3.9) with g replaced by $\nabla_x L(x,\lambda)$ and (4.8)) are used when suitable constraint scalings are available (see Section 5.1). In all other cases, the tests (4.6) and (4.7) are used.

4.2 Minimizing the augmented Lagrangian function

The convergence of augmented Lagrangian methods is guaranteed, under very weak assumptions, if the penalty parameter is gradually reduced to zero. This result is almost independent of the values of the Lagrange multiplier estimates (see [13, Lemma 4.3 and Theorem 4.4]). However, it becomes more difficult to minimize (4.4) when $\mu^{(k)}$ is small. Fortunately, a judicious choice of the Lagrange multiplier estimates also ensures convergence for *fixed* μ provided $x^{(k)}$ is close to x^*. Thus $\mu^{(k)}$ is allowed to decrease until we are sure that we are in a neighbourhood of x^* whereupon $\mu^{(k)}$ is left unchanged but the Lagrange multipliers are adjusted to ensure ultimate convergence. We can gauge whether we are in such a neighbourhood by monitoring the expected decrease in $\|c(x^{(k)})\|$.

At each iteration, we exit from SBMIN when the condition

$$\|x^{(k+1)} - P(x^{(k+1)} - \nabla_x \Phi(x^{(k+1)}, \lambda^{(k)}, S^{(k)}, \mu^{(k)}), l, u)\| \leq \omega^{(k)} \qquad (4.9)$$

is satisfied for some tolerance $\omega^{(k)}$. We then test whether

$$\|c(x^{(k+1)})\| \leq \eta^{(k)} \tag{4.10}$$

If (4.10) is satisfied, we leave the penalty parameter unchanged but update the Lagrange multiplier estimates. Otherwise, we reduce the penalty parameter and do not update the Lagrange multiplier estimates.

4.3 Updates

It is straightforward to pick $\mu^{(k)}$, $\lambda^{(k)}$, $\omega^{(k)}$ and $\eta^{(k)}$ to ensure convergence of the above scheme. Moreover this can normally be done so as to guarantee that the penalty parameter remains bounded away from zero (see [13, Theorem 5.5]).

We start by selecting positive tolerances $\tau = 0.1$, $\alpha_\omega = 1$, $\beta_\omega = 1$, $\alpha_\eta = 0.1$ and $\beta_\eta = 0.9$. Now, we set

$$\begin{array}{rcl} \mu^{(0)} & = & \mu_0 \\ \omega^{(0)} & = & \omega_0 \left(\mu^{(0)}\right)^{\alpha_\omega} \\ \eta^{(0)} & = & \eta_0 \left(\mu^{(0)}\right)^{\alpha_\eta}. \end{array} \tag{4.11}$$

We also pick initial Lagrange multiplier estimates $\lambda^{(0)}$ and scaling matrix $S^{(0)}$. In the absence of a better choice, LANCELOT selects $\lambda^{(0)} = 0$ and $S^{(0)} = I$.

The parameters are updated in different ways depending upon whether or not (4.10) is satisfied. When (4.10) holds, the next vector of Lagrange multiplier estimates (often known as first order multiplier estimates) is chosen as

$$\lambda^{(k+1)} = \lambda^{(k)} + S^{(k)} c(x^{(k)})/\mu^{(k)}. \tag{4.12}$$

The remaining parameters are given by

$$\begin{array}{rcl} \mu^{(k+1)} & = & \mu^{(k)} \\ \omega^{(k+1)} & = & \omega^{(k)} \left(\mu^{(k+1)}\right)^{\beta_\omega} \\ \eta^{(k+1)} & = & \eta^{(k)} \left(\mu^{(k+1)}\right)^{\beta_\eta}. \end{array} \tag{4.13}$$

When (4.10) does not hold, the penalty parameter is reduced. We set

$$\begin{array}{rcl} \mu^{(k+1)} & = & \tau \mu^{(k)} \\ \omega^{(k+1)} & = & \omega_0 \left(\mu^{(k+1)}\right)^{\alpha_\omega} \\ \eta^{(k+1)} & = & \eta_0 \left(\mu^{(k+1)}\right)^{\alpha_\eta}. \end{array} \tag{4.14}$$

The Lagrange multiplier estimates are left unchanged so that $\lambda^{(k+1)} = \lambda^{(k)}$.

Notice that, if (4.12) holds,

$$\nabla_x \Phi(x^{(k+1)}, \lambda^{(k)}, S^{(k)}, \mu^{(k)}) \equiv \nabla_x L(x^{(k+1)}, \lambda^{(k+1)}). \tag{4.15}$$

In this case, the overall convergence tests (4.6) and (4.7) are satisfied at the start of iteration $k+1$ provided that $\omega^{(k)} \leq \epsilon_l$ and $\eta^{(k)} \leq \epsilon_c$.

Automatic choices of the initial penalty parameter in penalty function methods are notoriously hard to justify. On the other hand, the choice is less critical if the constraints are well scaled. By default, LANCELOT selects $\mu_0 = 0.1$; this may be overruled by the user. We also choose $\omega_0 = 1$ and $\eta_0 = 0.1258925$ (thus $\eta^{(0)} = 0.01$).

5 Algorithmic options within LANCELOT

5.1 Constraint and variable scaling

In almost all that we have said up until now, there has been an implicit assumption that the values of the problem variables are all typically the same size. Similarly, it has been assumed that all the constraint functions are of similar magnitude. It may happen in practice that this is not so. The convergence of the method may be adversely affected by poor scaling and it should be avoided if at all possible. There are two alternatives. A user may manually rescale the variables and constraints so that the scaled quantities are of roughly the same size. Alternatively, the user might rely on an automatic rescaling algorithm.

There has been a fair amount written on scaling (see, for example, [31, Section 8.7] and [22, Section 7.1], and there is some consensus that it is extremely difficult to design a general purpose automatic scheme, especially for highly nonlinear problems. Notwithstanding, we still feel that such a scheme should be provided as an option.

LANCELOT allows the user to specify variable and constraint scalings as input parameters and the scalings are then used implicitly by the algorithms. It is also possible to construct automatic scalings independent of the minimization routines as follows.

Consider the vector valued function

$$v(x) = \begin{pmatrix} c_1(x) \\ \vdots \\ c_m(x) \\ f(x) \end{pmatrix}. \tag{5.1}$$

Let $F(x)$ denote the Jacobian matrix, $F_{ij}(x) = \delta v_i(x)/\delta x_j$. This matrix reflects the changes in the elements of v which are likely for small identical changes in x. If we were to change to variables $\hat{x} = D_x x$ and rescale v as $\hat{v} = D_v v$, where D_x and D_v are positive definite and diagonal, the Jacobian matrix of $\hat{v}$ with respect to the variables $\hat{x}$ is

$$D_v F(x) D_x^{-1}. \tag{5.2}$$

Ideally, we would like to choose the scalings so that the rows and columns of (5.2) are of roughly equal norm for all x in the feasible box (3.2). However, this is in general impossible for nonlinear functions and we must accept a compromise.

Let $x^{typical}$ be a typical value of x within the feasible box. We now apply the matrix equilibration algorithm of Curtis and Reid [20] to $F(x^{typical})$ to derive suitable scaling matrices D_x and D_v

to equilibrate (5.2). The first m diagonals of D_v are rescaled by the constant $1/\max_{1\leq i\leq m}(D_v)_{ii}$. The resulting matrices D_x and the first m components of the rescaled D_v are now passed as input parameters to the minimizer. The Curtis-Reid algorithm is implemented as MC19 in the Harwell Subroutine Library. This automatic scaling procedure is available as an option within LANCELOT and will be referred to as the "scaling" option.

5.2 Linear solvers

Most of the LANCELOT algorithmic options are related to the way in which an (approximate) minimizer of (3.25) is computed. This is hardly surprising since one expects the burden of the numerical calculation to be at this level.

5.2.1 Direct methods

Once the set $\mathcal{I}^{(k,j)}$ determined, the nature of the quadratic model restricted to the subset of free variables is characterized by the eigenvalue distribution (or inertia) of the matrix $\bar{B}^{(k,j)}$. To summarize:

- If all the eigenvalues of $\bar{B}^{(k,j)}$ are strictly positive, the unique minimizer of (3.25) is given as the solution to the Newton equations (3.26).
- If all the eigenvalues of $\bar{B}^{(k,j)}$ are nonnegative, but some are zero, and $\bar{g}^{(k,j)}$ lies in the range of $\bar{B}^{(k,j)}$, there are an infinite number of solutions to the Newton equations (3.26). Each solution is a weak minimizer of the model.
- If all the eigenvalues of $\bar{B}^{(k,j)}$ are nonnegative, but some are zero, while $\bar{g}^{(k,j)}$ does not lie in the range of $\bar{B}^{(k,j)}$, the quadratic model is unbounded from below. There is then a vector $\bar{p}^{(k,j)}$ for which $\bar{g}^{(k,j)T}\bar{p}^{(k,j)} < 0$ and $\bar{B}^{(k,j)}\bar{p}^{(k,j)} = 0$. This vector is known as a *direction of linear infinite descent* (dolid) since the model decreases as a linear function of α for steps $\alpha\bar{p}^{(k,j)}$ as α increases.
- If $\bar{B}^{(k,j)}$ has negative eigenvalues, the model is unbounded from below. There is a vector $\bar{p}^{(k,j)}$ for which $\bar{g}^{(k,j)T}\bar{p}^{(k,j)} \leq 0$ and $\bar{p}^{(k,j)T}\bar{B}^{(k,j)}\bar{p}^{(k,j)} < 0$. This vector is known as a *direction of negative curvature* (donc). The model decreases as a quadratic function of α for steps $\alpha\bar{p}^{(k,j)}$ as α increases.

The use of a sparse multifrontal direct method to solve large-scale optimization problems has been advocated in [9]. Briefly, the matrix $\bar{B}^{(k,j)}$ is factorized using the Harwell Subroutine Library code MA27 [25], [26] as

$$\bar{B}^{(k,j)} = \bar{\Pi}^{(k,j)}\bar{L}^{(k,j)}\bar{D}^{(k,j)}\bar{L}^{(k,j)T}\bar{\Pi}^{(k,j)T}, \tag{5.3}$$

where $\bar{\Pi}^{(k,j)}$ is a permutation matrix, $\bar{L}^{(k,j)}$ is unit lower triangular and $\bar{D}^{(k,j)}$ is block diagonal with 1 by 1 and 2 by 2 diagonal blocks. The inertia of $\bar{B}^{(k,j)}$ and $\bar{D}^{(k,j)}$ are identical.

A first option within LANCELOT, denoted by the "mltf" symbol, uses the multifrontal factorization directly to find the Newton direction when $\bar{B}^{(k)}$ is positive definite. It uses its computed triangular factors to calculate a suitable direction $\bar{p}^{(k,j)}$ in the three other cases, as is detailed in [9]. We allow only one cycle of improvement beyond the Cauchy point with this option, that is j is limited to 1 in Section 3.3.

An alternative direct method which shares the key property that the Newton direction is always chosen if $\bar{B}^{(k,j)}$ is positive definite is based on the modified Cholesky methods of Schnabel and Eskow ([52]). Here, we form a factorization

$$\bar{B}^{(k,j)} + \bar{E}^{(k,j)} = \bar{L}^{(k,j)} \bar{D}^{(k,j)} \bar{L}^{(k,j)T}, \tag{5.4}$$

where $\bar{L}^{(k,j)}$ is unit lower triangular, $\bar{D}^{(k,j)}$ is positive definite and diagonal, and $\bar{E}^{(k,j)}$ is positive semi-definite, diagonal and nonzero only when $\bar{B}^{(k,j)}$ is not (sufficiently) positive definite. It is straightforward to modify MA27 to achieve this factorization. Now, the modified Newton equations,

$$(\bar{B}^{(k,j)} + \bar{E}^{(k,j)})\bar{p}^{(k,j)} = -\bar{g}^{(k,j)} \tag{5.5}$$

are solved to obtain a suitable search direction. Notice, furthermore, that

$$\begin{aligned} \bar{m}^{(k,j)}(\bar{p}^{(k,j)}) &\leq m^{(k)}(x^{(k,j)}) + \bar{g}^{(k,j)T}\bar{p}^{(k,j)} + \frac{1}{2}\bar{p}^{(k,j)T}(\bar{B}^{(k,j)} + \bar{E}^{(k,j)})\bar{p}^{(k,j)} \\ &\leq \bar{m}^{(k,j)}(\bar{p}^{(k,1)}). \end{aligned} \tag{5.6}$$

In contrast with the mltf option, more than one cycle of improvement beyond the Cauchy point is allowed with this latter option, which will be denoted below by the symbol "semltf".

We stress that an advantage of both these techniques is that $B^{(k)}$ will typically not be modified as we approach the solution to the problem. Moreover, provided the trust-region radius is sufficiently large that the Newton step (3.26) may be taken, we would also expect to take very few inner-iterations (indeed, in the nondegenerate case, one) before (3.23) is satisfied.

5.2.2 Iterative methods

There is sometimes a very fine distinction between iterative and direct methods. In fact, in finite precision arithmetic, one is often advised to perform iterative refinement with direct methods to obtain more accurate solutions. Iterative methods are in general more flexible in the sense that it may be impossible to use direct methods because of insufficient storage on a user's machine, while iterative methods can be adapted to use whatever space is available. However,this flexibility has its drawbacks. In particular, the convergence of iterative methods on difficult problems can be severely impaired unless considerable care is taken.

In LANCELOT, the iterative method of choice is the method of conjugate gradients (see, for example, [31, Section 4.8.3], or [33, Sections 10.2 and 10.3]. Such a method attempts to find a stationary point of a quadratic function, in our case (3.25), by generating a sequence of (conjugate) search directions, $\bar{p}^{(k,j)}$. If $\bar{B}^{(k,j)}$ is not positive definite, the conjugate gradients may terminate with a donc or dolid.

The convergence of the conjugate gradient method may be enhanced by preconditioning the coefficient matrix $\bar{B}^{(k,j)}$. A preconditioner is a symmetric, positive definite matrix $\bar{P}^{(k,j)}$ which is chosen to make the eigenvalues of the product $\bar{P}^{(k,j)-1}\bar{B}^{(k,j)}$ cluster around as few distinct values as possible. If $\bar{B}^{(k,j)}$ were positive definite, the ideal preconditioner would be $\bar{B}^{(k,j)}$ itself. However, we have to bear in mind that at each step of the preconditioned conjugate gradient method we have to solve linear equations of the form

$$\bar{P}^{(k,j)}\bar{z}^{(k,j)} = -\bar{r}^{(k,j)}, \tag{5.7}$$

for given vectors $\bar{r}^{(k,j)}$ and required solutions $\bar{z}^{(k,j)}$. Thus there is normally a compromise between using a good approximation of $\bar{B}^{(k,j)}$, with the associated difficulties of finding and storing its factorization, and a poor approximation, where many conjugate gradient iterations may be required. Choosing a good preconditioner for a given problem is considered to be an art. Certain classes of problems, in particular those associated with fluid flows, have been much analyzed and reasonable preconditioners designed.

We have tried to supply a reasonable cross-section of widely used preconditioners. We recognize that users may have a better idea of a good preconditioner for their problem by allowing them to solve (5.7) outside the package.

Diagonal Preconditioners. The choice $\bar{P}^{(k,j)} = I$ is, of course, a possibility. This will be referred to as the "noprc" option. A simple but often useful preconditioner is provided by the choice $\bar{P}^{(k,j)} = \bar{D}^{(k,j)}$, where $\bar{D}^{(k,j)}$ is a diagonal matrix whose i-th diagonal is

$$\bar{D}_{ii}^{(k,j)} = \max(\bar{B}_{ii}^{(k,j)}, \epsilon), \tag{5.8}$$

for some suitable small number ϵ. We set ϵ to the cube root of the machine precision.

We note that diagonal preconditioners can be used to precondition both the steepest descent direction in the Generalized Cauchy point calculation ($D^{(k)} = \bar{D}^{(k,j)}$ in (3.10)) and within the conjugate gradient iteration itself. We will denote by diagonal a preconditioning stategy using this technique. In this strategy, the trust region radius is furthermore scaled by the factor $\sqrt{\|\bar{D}^{(k,j)}\|}$, as suggested by the theory [10].

Band Preconditioners. Many application areas give rise to problems whose Hessian matrices are banded. A band matrix is a matrix B for which $b_{ij} = 0$ for all $|i-j| \leq m_b$. The smallest integer m_b for which this is so is known as the semi-bandwidth of the matrix. The significant property as far as we are concerned is that, if B is positive definite, the Cholesky factors fit within the band. Moreover, clever storage schemes have been constructed to make the factorization and subsequent solutions extremely efficient (see, for example, [28, Chapter 4], and [24, Section 10.2]). We offer a band preconditioner within LANCELOT. This works in two stages. The desired semi-bandwidth, m_b, is assumed to have been specified. The band matrix $\bar{M}^{(k,j)}$, with semi-bandwidth m_b, is chosen so that

$$\bar{M}_{il}^{(k,j)} = \bar{B}_{il}^{(k,j)} \text{ for all } |i-l| \leq m_b. \tag{5.9}$$

Then, we obtain a modified Cholesky factorization of $\bar{M}_{il}^{(k,j)}$, just as described in Section 5.2.1. That is we form a factorization

$$\bar{P}^{(k,j)} \equiv \bar{M}^{(k,j)} + \bar{E}^{(k,j)} = \bar{L}^{(k,j)}\bar{D}^{(k,j)}\bar{L}^{(k,j)T}, \tag{5.10}$$

where $\bar{L}^{(k,j)}$ is unit lower triangular with semi-bandwidth m_b, $\bar{D}^{(k,j)}$ is positive definite and diagonal, and $\bar{E}^{(k,j)}$ is positive semi-definite, diagonal and nonzero only when $\bar{M}^{(k,j)}$ is not (sufficiently) positive definite.

When $\bar{B}^{(k,j)}$ is positive definite and m_b is chosen large enough, $\bar{P}^{(k,j)}$ is an exact preconditioner, that is, the preconditioned conjugate gradient method will converge in a single iteration. The effect of the preconditioner in other cases has not been formally analyzed. However, one would suspect that if the essential part of $\bar{B}^{(k,j)}$ is contained within the band, the preconditioner would be successful.

Band preconditioners are denoted below by "band(m_b)".

Incomplete Factorization Preconditioners. In some applications, although it is possible to store the matrix $\bar{B}^{(k,j)}$, there is so much fill-in during the factorization (5.3) that it proves impossible to store the resulting factor $\bar{L}^{(k,j)}$. Such cases arise frequently if the underlying problem has connections with two and, particularly, three dimensional partial differential equations.

It is sometimes possible to construct good preconditioners for such problems by either rejecting all fill-in during the factorization or by tolerating a modest amount. Such incomplete factorization preconditioners are very popular with researchers in partial differential equations and it is possible to get off-the-shelf software to form them. We include the example MA31, due to Munksgaard [47], from the Harwell Subroutine Library in LANCELOT. We denote this option by "munksg".

Full-Matrix Preconditioners. Finally, as we alluded to in Section 5.2.2, if space permits and $\bar{B}^{(k,j)}$ is positive definite, one can always use a complete factorization of $\bar{B}^{(k,j)}$ as a preconditioner. However, if $\bar{B}^{(k,j)}$ is not positive definite it is possible to use one of the modifications suggested in Section 5.2.1 to determine a preconditioner. In fact, we allow the modification (5.4).

We consider two possible ways to obtain the perturbation matrix $\bar{E}^{(k,j)}$ in (5.5). The first is, as above, the modified factorization algorithm proposed by Schnabel and Eskow in [52]. We will use "seprc" to denote this strategy.

The second is another modification of MA27 advocated by Gill, Murray, Ponceléon and Saunders in [30]. Here, rather than modifying the factorization as it is being formed, the factorization (5.3) is computed and *then* modified. Specifically, the block diagonal matrix $\bar{D}^{(k,j)}$ in (5.3) is changed to $\bar{D}^{(k,j)} + \bar{E}^{(k,j)}$. This modified matrix is block diagonal positive definite. This ensure that

$$\bar{\Pi}^{(k,j)}\bar{L}^{(k,j)}(\bar{D}^{(k,j)} + \bar{E}^{(k,j)})\bar{L}^{(k,j)T}\bar{\Pi}^{(k,j)T} \tag{5.11}$$

is also positive definite. Once again, the matrix is not modified when $\bar{B}^{(k,j)}$ is positive definite. The resulting algorithmic option is denoted below by "gmpsprc".

It is worthwhile noting the parallel between seprc and semltf. They both use the direct modified factorization of $\bar{B}^{(k,j)}$ to compute Newton's direction in the subspace of free variables. They differ in we decided to stop this process in seprc as soon as the only bounds encountered are trust region bounds, while the minimization may be pursued, in semltf, along the trust region boundaries.

The variants gmpsprc and mltf are also related, because they both use the MA27 multifrontal factorization. The gmpsprc then modifies it if necessary and uses the resulting factors to precondition a conjugate gradient minimizer, while mltf directly exploits the factors to compute a single step to reduce the quadratic model.

Expanding Band Preconditioners. One further possibility is to use an expanding band preconditioner. Consider the band matrix $\bar{M}^{(k,j)}$ given by (5.9), where the semi-bandwidth m_b satisfies the inequalities $0 \leq m_{bmin} \leq m_b \leq m_{bmax} \leq n$. Here m_{bmax} is some upper bound on the allowable bandwidth chosen so that a sparse factorization of $\bar{M}^{(k,j)}$ is practicable. The lower bound m_{bmin} is selected so that the band matrix with exactly that semi-bandwidth provides a useful preconditioner. (Of course, both of these bounds are difficult, if not impossible, to pick *a priori* and one might just choose $m_{bmin} = 0$ and $m_{bmax} = n$.) The idea is now to select the semi-bandwidth $m_b^{(k)}$ at each iteration to reflect the speed and accuracy which one wants from the preconditioned conjugate gradient method. In particular, if low accuracy is required, a preconditioner with a small semi-bandwidth (such as a diagonal preconditioner) is often very effective. But if high accuracy is desired, it may be better to pick a preconditioner which is a better approximation to $\bar{B}^{(k,j)}$.

Having obtained the preconditioner, we obtain a modified Cholesky factorization of $\bar{M}_{il}^{(k,j)}$, just as described in Section 5.2.1. However, unlike the band preconditioners described above, the matrix and its factorization are stored as a general sparse, rather than band, matrix.

We use the following very simple rule to select the semi-bandwidth. Pick

$$m_b^{(k)} = \begin{cases} n & \text{if } \|x^{(k)} - P(x^{(k)} - g(x^{(k)}), l, u)\| \leq 10^{-2}, \\ n/2 & \text{if } 10^{-2} < \|x^{(k)} - P(x^{(k)} - g(x^{(k)}), l, u)\| \leq 10^{-1}, \\ n/5 & \text{otherwise.} \end{cases} \tag{5.12}$$

We realize that further sophistication may be desirable but have found that this simple scheme is effective in practice. This preconditioning option will be denoted by "expband".

5.3 Derivative approximations

Further algorithmic options in LANCELOT are related to the various ways in which derivatives or their approximations are computed. However, the structure of these derivatives crucially depends on the structure of the nonlinear functions themselves. In order to derive an efficient algorithm for large-scale calculations, we first need to know a way to handle the structure typically inherent in functions of many variables.

5.3.1 Group partial separability

A function $f(x)$ is said to be *group partially separable* if:

1. the function can be expressed in the form

$$f(x) = \sum_{i=1}^{n_g} g_i(\alpha_i(x)); \tag{5.13}$$

2. each of the *group functions* $g_i(\alpha)$ is a twice continuously differentiable function of the single variable α;

3. the function
$$\alpha_i(x) = \sum_{j \in \mathcal{J}_i} w_{i,j} f_j(x^{[j]}) + a_i^T x - b_i \tag{5.14}$$
is known as the i-th *group*;

4. each of the index sets $\mathcal{J}_i$ is a subset of $\{1, \ldots, n_e\}$;

5. each of the *nonlinear element functions* f_j is a twice continuously differentiable function of a subset $x^{[j]}$ of the variables x. Each function is assumed to have a large invariant subspace. Usually, this is manifested by $x^{[j]}$ comprising a small fraction of the variables x;

6. the gradient a_i of each of the *linear element functions* $a_i^T x - b_i$ is, in general, sparse; and

7. the $w_{i,j}$ are known as element *weights*.

This structure is extremely general. Indeed, any function with a continuous, sparse Hessian matrix may be written in this form (see [34]). A more thorough introduction to group partial separability is given in [12]. LANCELOT assumes that the objective function $f(x)$ is of this form. When equality constraints are present, they are handled via the augmented Lagrangian and thus become part of the objective function for the subproblem given to SBMIN. Each such constraint then gives rise to the group function $\alpha^2/2\mu$, which imposes the restriction that each equality constraint only has a single group.

5.3.2 Derivatives and their approximations

One of the main advantages of the group partial separable structure is that it considerably simplifies the calculation of derivatives of $f(x)$. If we consider (5.13) and (5.14), we see that we merely need to supply derivatives of the nonlinear element and group functions. LANCELOT then assembles the required gradient and, possibly, Hessian matrix of f from this information.

5.3.3 Derivatives of $f(x)$

The gradient of (5.13) is given by

$$\nabla_x f(x) = \sum_{i=1}^{n_g} g_i'(\alpha_i(x)) \nabla_x \alpha_i(x), \tag{5.15}$$

where the gradient of the i-th group is

$$\nabla_x \alpha_i(x) = \sum_{j \in \mathcal{J}_i} w_{i,j} \nabla_x f_j(x^{[j]}) + a_i. \tag{5.16}$$

Similarly, the Hessian matrix of the same function is given by

$$\nabla_{xx} f(x) = \sum_{i=1}^{n_g} g_i''(\alpha_i(x)) \nabla_x \alpha_i(x) (\nabla_x \alpha_i(x))^T + \sum_{i=1}^{n_g} g_i'(\alpha_i(x)) \nabla_{xx} \alpha_i(x), \tag{5.17}$$

where the Hessian matrix of the i-th group is

$$\nabla_{xx}\alpha_i(x) = \sum_{j \in \mathcal{J}_i} w_{i,j} \nabla_{xx} f_j(x^{[j]}). \tag{5.18}$$

Notice that the Hessian matrix is the sum of two different types of terms. The first is a sum of rank-one terms only involving first derivatives of the nonlinear element functions. The second involves second derivatives of the nonlinear elements.

We shall assume that the first and second derivatives of the group functions are available. This is frequently the case in practice.

The quadratic model (3.3) uses the gradient of f and an approximation to its Hessian matrix. In LANCELOT, we have two options.

- We can calculate the true first and second derivatives of each nonlinear element and group function and use the exact Hessian $B^{(k)} = \nabla_{xx} f(x^{(k)})$.
- We can calculate the true first and second derivatives of each group function, calculate the first derivatives of the nonlinear elements but use approximations, $B_i^{[j](k)}$, to their second derivatives. We then use the approximation

$$B^{(k)} = \sum_{i=1}^{n_g} g_i''(\alpha_i(x^{(k)})) \nabla_x \alpha_i(x^{(k)}) (\nabla_x \alpha_i(x^{(k)}))^T + \sum_{i=1}^{n_g} g_i'(\alpha_i(x^{(k)})) B_i^{(k)}, \tag{5.19}$$

where $B_i^{(k)}$ satisfies

$$B_i^{(k)} = \sum_{j \in \mathcal{J}_i} w_{i,j} B^{[j](k)} \tag{5.20}$$

for some suitable matrices $B^{[j](k)}$, see Section 5.3.5.

We strongly recommend the use of exact second derivatives whenever they are available. LANCELOT fully exploits this information. In our experience, exact second derivatives are often available, either by direct calculation or by using automatic differentiation tools. Using exact second derivatives is therefore the default option in the package.

5.3.4 Derivatives of the group and nonlinear element functions

In order to describe the other options for derivative approximation, we need to detail the structure of these further.

The derivatives of f_j need only be found with respect to the variables $x^{[j]}$, the remaining derivatives being zero. There are frequently, moreover, two further savings to be made. Firstly, although the variables used by an individual f_j may differ, the structure of many of the nonlinear elements may be the same. For instance, $f_1(x^{[1]})$ might be $x_1 x_2$ and $f_2(x^{[2]})$ might be $x_3 x_4$; both functions are of the generic *type* $e(v_1, v_2) = v_1 v_2$. If we need the derivatives of these two nonlinear elements, all we have to compute are the derivatives of the generic function $e(v_1, v_2)$ and to associate the variables x_1 and x_2 and x_3 and x_4 with v_1 and v_2, respectively (A more realistic example is described in [12]). In general, we associate each nonlinear element function

f_j, $1 \leq j \leq n_e$ with a generic type of nonlinear function from the set $\{e_m(v^{[m]})\}$, $1 \leq m \leq m_e$, via a suitable mapping $m(j)$. We then only need to know the derivatives of the different types of nonlinear element functions and remember which variables $x^{[j]}$ are associated with the relevant *elemental variables* $v^{[m]}$. Secondly, computing the derivatives of a given type of nonlinear element function may be further simplified. For example, suppose that a nonlinear element is of type $e(v_1, v_2, v_3) = ((v_1 + v_2)(v_2 - v_3))^2$. Although e is a function of three elemental variables, it only really depends on two internal variables $u_1 = v_1 + v_2$ and $u_2 = v_2 - v_3$; in this case, $e(v_1, v_2, v_3) = \hat{e}(u_1, u_2) = (u_1 u_2)^2$. The nonlinear information is still conveyed in $\hat{e}$. The derivatives with respect to the variables v may be calculated by computing the derivatives with respect to the variables u and then using the chain rule of partial differentiation to recover the required ones.

In general, we might obtain *internal* variables $u^{[m]}$ from the elemental variables $v^{[m]}$ by the linear *range* transformation

$$u^{[m]} = W^{[m]} v^{[m]}. \tag{5.21}$$

The transformation is only *useful* if the transformation matrix $W^{[m]}$ has fewer rows than columns. Note that the transformation is far from unique and, in many cases, there is no useful transformation (in which case the trivial transformation, with $W^{[m]} = I$, suffices). The gradient and Hessian matrix of a nonlinear element type $e_m(v^{[m]}) \equiv \hat{e}_m(u^{[m]})$, with respect to the elemental variables, may be calculated from those with respect to the internal variables as

$$\nabla_v e_m(v^{[m]}) = W^{[m]T} \nabla_u \hat{e}_m(u^{[m]}) \tag{5.22}$$

and

$$\nabla_{vv} e_m(v^{[m]}) = W^{[m]T} \nabla_{uu} \hat{e}_m(u^{[m]}) W^{[m]}. \tag{5.23}$$

Thus, in order to calculate these derivatives, we have merely to calculate the derivatives with respect to the internal variables and record the range transformations made. The remaining part of the calculation is automated within LANCELOT.

5.3.5 Approximating the derivatives of nonlinear element functions

We are now in position to describe these LANCELOT options more precisely.

We see in (5.22) that the nonlinear information in the first derivatives of the nonlinear element function $e_m(v^{[m]}) \equiv \hat{e}_m(u^{[m]})$ is conveyed by the vector $\nabla_u \hat{e}_m(u^{[m]})$. If we seek an approximation $g^{[j](k)}$ to $\nabla_x f_j(x^{[j](k)})$ in (5.16), we should first determine the type of nonlinear element j, $m \equiv m(j)$ say. Now we approximate $\nabla_u \hat{e}_m(x^{[j](k)})$ by a suitable vector $\hat{g}^{[j](k)}$ and form

$$g^{[j](k)} = W^{[m]T} \hat{g}^{[j](k)}. \tag{5.24}$$

The approximations $\hat{g}^{[j](k)}$ would normally be formed by finite differences. As the dimension of $u^{[m]}$ is assumed to be small (f_j has a large invariant subspace), this is realistic even for large problems.

A finite difference approximation to the i-th component of the required gradient is calculated by forward differences

$$\frac{\hat{e}_m(x^{[j](k)} + \delta_i e_i) - \hat{e}_m(x^{[j](k)})}{\delta_i} \tag{5.25}$$

until the projected gradient of the objective function is small when a switch is made to central differences

$$\frac{\hat{e}_m(x^{[j](k)} + \delta_i e_i) - \hat{e}_m(x^{[j](k)} - \delta_i e_i)}{2\delta_i.} \tag{5.26}$$

In (5.25) and (5.26), δ_i is a suitable small positive number. Further details of such procedures are given in [22] and [31]. This option will be denoted by "**fdg**".

Turning to second derivatives, we see in (5.23) that the nonlinear information in the second derivatives of the nonlinear element function $\hat{e}_m(u^{[m]})$ is contained in the matrix $\nabla_{uu}\hat{e}_m(u^{[m]})$. As before, if we seek an approximation $B^{[j](k)}$ to $\nabla_{xx}f_j(x^{[j](k)})$ in (5.20), we should first determine the type of nonlinear element j, $m \equiv m(j)$ say. Now we approximate its Hessian $\nabla_{uu}\hat{e}_m(x^{[j](k)})$ by a suitable matrix $\hat{B}^{[j](k)}$ and form

$$B^{[j](k)} = W^{[m]T}\hat{B}^{[j](k)}W^{[m]}. \tag{5.27}$$

The approximations $\hat{B}^{[j](k)}$ may be formed by secant updating formulae, using information from previous iterations. As the dimension of $u^{[m]}$ is assumed to be small (f_j has a large invariant subspace), this option is realistic even for large problems. This technique is known as partitioned updating [35].

A secant approximation is any matrix chosen to satisfy the secant equation

$$\hat{B}^{[j](k)}\hat{s}^{[j](k)} = \hat{y}^{[j](k)}, \tag{5.28}$$

where

$$\hat{s}^{[j](k)} = W^{[m]}(x^{[j](k)} - x^{[j](k-1)}) \tag{5.29}$$

and

$$\hat{y}^{[j](k)} = \nabla_u\hat{e}_m(x^{[j](k)}) - \nabla_u\hat{e}_m(x^{[j](k-1)}). \tag{5.30}$$

The matrix is normally chosen to be symmetric. Sometimes, positive definiteness is also imposed, although there is little justification for this in our circumstances since there is no a priori reason for the second derivatives of the element functions to be positive definite.

LANCELOT presently uses the same type of derivative approximation for all elements. The Powell-symmetric-Broyden (PSB), symmetric-rank-one (SR1), the Broyden-Fletcher-Goldfarb-Shanno (BFGS) and Davidon-Fletcher-Powell (DFP) updates are provided. These choices are referred to as the "**psb**", "**sr1**", "**bfgs**" and "**dfp**" options respectively. See [22], [27] and [31] for further details on these updating formulae.

5.4 Other options within SBMIN

We finally briefly described three further options related to the algorithms used within SBMIN.

5.4.1 Approximate Cauchy point calculation

In Section 3.2, we distinguished between two types of Cauchy point calculations. The first is to compute the first local minimum of the quadratic model along the Cauchy arc (3.11). This is the default choice. The second is to compute an approximate Cauchy point, an option that we will denote by "**appGCP**".

5.4.2 Choice of the trust region norm

The next algorithmic option that we will consider is that of the choice of the norm $\|\cdot\|$. This choice determines the actual shape of the trust region, because of (3.4) and (3.13). The default option is to choose $\|\cdot\|$ to be the infinity norm, but the choice of the Euclidean ℓ_2-norm is also allowed. Note that in this latter case the boundary of the region defined as the intersection of the trust region (3.4) and the bound constraints (3.2) has a more complex shape than when the infinity norm is used. This prevents one from using the technique of [11] for determining the Generalized Cauchy point on the boundary of this more complex region. When the ℓ_2-norm is selected, LANCELOT therefore stops the search for the GCP as soon as the trust region boundary is encountered along the Cauchy arc. We will denote this latter option by "l2norm".

5.4.3 Accurate solution of the BQP

Finally, the last option allows the user to specify that the minimization of the objective function model has to be accurate within the intersection of the feasible region for the bound constraints and the trust region. In Section 3.3, we gave a general framework for obtaining a new iterate that is "better" than the generalized Cauchy point. At each stage, an approximation to the minimizer of the model is sought while some of the variables are held fixed at bounds. This set of fixed variables, $\mathcal{I}^{(k,j)}$, always includes those which were fixed at the approximation to the generalized Cauchy point. In SBMIN, we also include by default all variables which encounter bounds at $x^{(k,j)}$, for $j > 0$ up until the test (3.23) is satisfied. Then, optionally, we may free all variables except those which were fixed at the approximation to the generalized Cauchy point and perform one or more further cycles. This optional process, denoted by "accBQP", is terminated when releasing variables does not improve the model value. This is detected when (3.23) and

$$Q(\nabla_x m^{(k)}(x^{(k,j)}), x^{(k,j)}, l, u) = Q(\nabla_x m^{(k)}(x^{(k,j)}), x^{(k,1)}, l, u) \tag{5.31}$$

are satisfied. At the start of each cycle, we also compute a new generalized Cauchy point for the model fixing the variables which were on a bound at the original Cauchy point. This recursive use of SBMIN is guaranteed to satisfy (3.23) if a sufficient number of cycles are performed.

6 The numerical tests: framework and procedure

6.1 The test problems

The numerical tests with LANCELOT that we are about to describe were conducted using the Constrained and Unconstrained Test Examples (CUTE) collection of nonlinear test problems (see [3]). This collection contains a large number of nonlinear optimization problems of various sizes and difficulty, representing both "academic" and "real world" applications. As the title of the collection implies, constrained and unconstrained examples are included. For our tests, we have used the first 624 instances of unconstrained (or bound constrained) problems and the first 319 instances of constrained problems. In the CUTE collection, these are numbered U1 to U624 and C1 to C324 respectively[1]. These 943 instances are derived from 398 *different* problems, the additional examples being determined by varying the dimension. It is of course undesirable to describe all these examples in the present paper. It will suffice to say that our test set covers, amongst others,

- the "Argonne test set" [45], the Testpack report [5], the Hock and Schittkowski collection [38], the Dembo network problems (see [21]), the Moré-Toraldo quadratic problems [46], the Toint-Tuyttens network model problems [59],
- most problems from the PSPMIN collection [56][2],
- problems inspired by the orthogonal regression report by Gulliksson [36],
- some problems from the Minpack-2 test problem collection[3] [1], [2] and from the second Schittkowski collection [50],
- a number of original problems from various application areas.

We present some of the problems characteristics in Figures 3 and 4 and in Table 1.

- Figure 3 shows the distribution of the problems' *dimensions*.
- Figure 4 illustrates the distribution of the ratio m/n, where m is the total number of general equality and inequality constraints. The higher this ratio, the "more constrained" the problem. Only constrained problems ($m > 0$) are considered in this statistic.
- Table 1 reports the number of our test problems with characteristics falling into five different classes. The characteristics considered are
  - the *relative nonlinearity of the objective function*, that is the ratio

    $$\nu_{\text{obj}} \overset{\text{def}}{=} \frac{\text{number of nonlinear groups in the objective}}{\text{number of groups in the objective}}, \tag{6.1}$$

    where the groups are defined in (5.14) and where a group is declared nonlinear if it contains at least one nontrivial nonlinear element function with a nonzero weight;
  - the *relative nonlinearity of the constraints*, i.e.

    $$\nu_{\text{cons}} \overset{\text{def}}{=} \frac{\text{number of nonlinear constraints}}{\text{number of constraints}}, \tag{6.2}$$

    where the bounds have been excluded from the denominator;
  - the proportion n_b/n of variables subject to *bound constraints*;
  - the *proportion of equality constraints*, that is of the ratio

    $$\gamma \overset{\text{def}}{=} \frac{\text{number of equality constraints}}{m}. \tag{6.3}$$

[1] It was supposed a priori that C17, C18 and C19 would excede our limitation on execution time, while C261 and C262 were excluded because of their non-differentiable behaviour.

[2] Some trivial problems were skipped and also problems for which different local minima were known.

[3] The problems that we could reconstruct from the data given in the report.

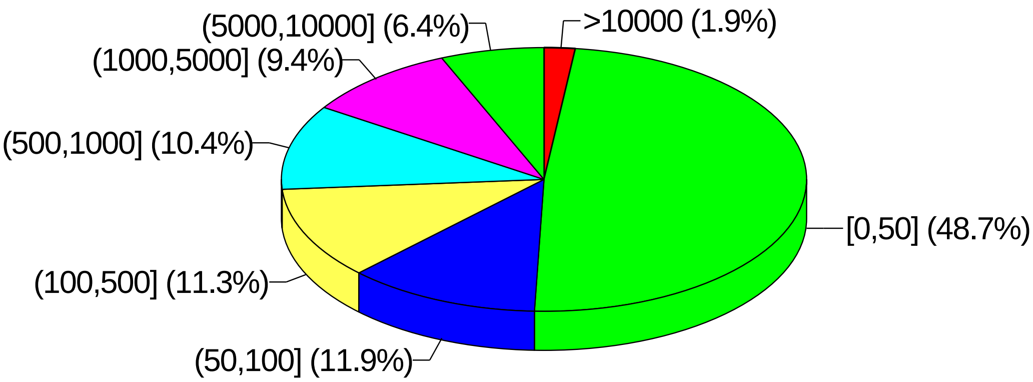

Figure 3: Distribution of problems dimension

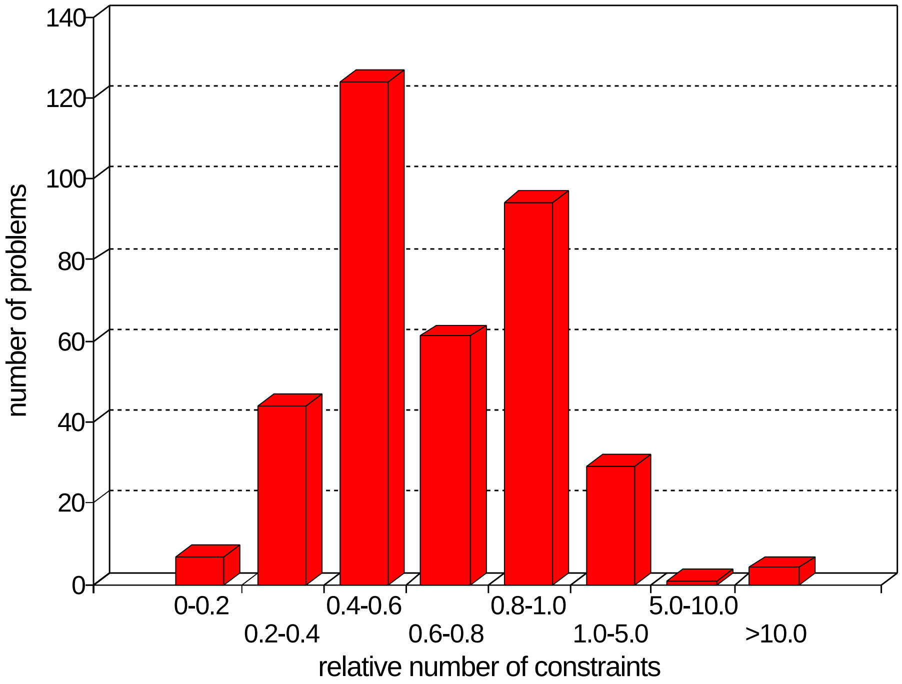

Figure 4: Distribution of the relative number of constraints m/n

	$[0, \frac{1}{5}]$	$(\frac{1}{5}, \frac{2}{5}]$	$(\frac{2}{5}, \frac{3}{5}]$	$(\frac{3}{5}, \frac{4}{5}]$	$(\frac{4}{5}, 1]$
ν_{obj}	104	8	31	16	784
ν_{cons}	139	5	20	8	192
n_b/n	573	25	38	14	293
γ	99	5	7	13	240

Table 1: Further problems characteristics

We note the following points.

- The majority of the problems are not very large. However, the classes of larger problems are far from empty, and we note the presence of examples with more than 15000 variables.
- Most large problems tend to have a somewhat regular structure. As a result, most groups in these problems tend to be structurally similar. This is noticeable in the distribution of relative nonlinearity of objective function and constraints, where either most or very few, if any, groups are nonlinear. The same phenomenon is also observed for the proportion of bounded variables which tends to be either very low or close to one.
- There are very few problems involving considerably more general constraints than variables. Many of the problems arise as nonlinear systems of equations, while a fair proportion have approximately half as many constraints as variables. We nevertheless note the presence of problems where the number of constraints is substantially greater than n.

Amongst the 943 tests problems, we also selected a subset of 45 large scale cases. These problems and their characteristics are presented in Table 2. In this Table, "nbr" stands for the number of the problem in the CUTE collection, n_{fr} the number of free variables, n_{fx} the number of fixed variables, n_{bd} the number of variables with bound constraints. The total number of optimization variables is therefore $n_{fr} + n_{bd}$. The symbol n_{lo} denotes the number of linear groups within the objective function, while n_{no} is that of nonlinear groups. The number of linear equality constraints is n_{le} while that of nonlinear ones is n_{ne}. Finally, n_{li} is the number of linear inequality constraints and n_{ni} that of nonlinear ones. The Table also provides a note indicating the relevant application domain or some feature of interest and a reference for the previously published problems.

It will be interesting to compare the performance of the package on the complete set of problems and on the selected subset of larger instances. Indeed this will allow refinement of our algorithmic conclusions for the large scale cases, where exploitation of problem's structure is of course crucial.

Name	nbr	n_{fr}	n_{fx}	n_{bd}	n_{lo}	n_{no}	n_{le}	n_{ne}	n_{li}	n_{ni}	Note	Ref.
ARTIF	U20	5000	2	0	0	5000	0	0	0	0	turning point	[39]
BIGBANK	C2	0	308	1922	0	1	1112	0	0	0	econometric model	[21]
BQPGAUSS	U46	0	0	2003	0	1	1	0	0	0	quadratic subproblem	
BRATU3D	U61	3375	1538	0	0	0	0	3375	0	0	Bratu PDE problem	[1]
BRIDGEND	C168	1423	0	1311	1	0	1304	1423	0	0	gas distribution	
BRITGAS	C3	0	0	450	0	1	0	360	0	0	gas distribution	
BROYDN3D	U73	10000	0	0	0	10000	0	0	0	0	Broyden tridiagonal	[45]
BROYDNBD	U84	5000	0	0	0	0	0	5000	0	0		[45]
CBRATU2D	U88	882	176	0	0	0	0	882	0	0	complex Bratu problem	[1]
CHEMRCTB	U116	0	0	1000	0	0	2	998	0	0	chemistry	[1]
CLPLATEC	U138	4970	71	0	1	19600	0	0	0	0	clamped plate problem	[49]
CORKSCRW	C299	247	9	200	0	50	300	0	0	50	optimal control	
DALLASL	C4	0	0	906	0	1	667	0	0	0	water distribution	[21]
DIXMAANE	U172	3000	0	0	0	4	0	0	0	0	Dixon and Maany problem	[23]
ENGVAL1	U232	5000	0	0	4999	4999	0	0	0	0	Engvall's problem	[56]
FREUROTH	U245	5000	0	0	0	9998	0	0	0	0	Freudenstein and Roth problem	[56]
GRIDNETB	C33	13284	0	0	0	1	6724	0	0	0	network optimization	[58]
HAGER4	C293	10000	1	0	0	10000	5000	0	0	0	optimal control	[37]
HYDROELL	C175	0	2	1007	0	1	0	0	1008	0	energy	[29]
JNLBRNGA	U307	0	496	15129	1	30752	0	0	0	0	engineering	[46]
LCH	C267	600	0	0	0	1	0	1	0	0	quantum physics	
LINVERSE	U624	999	0	1000	0	2997	0	0	0	0	matrix variational problem	
LMINSURF	U322	900	124	0	0	961	0	0	0	0	minimum surface	[56]
MANNE	C177	0	1	1094	0	1	0	0	365	365	econometry	[48]
MINPERM	C190	0	0	1113	1	0	20	1013	0	0	permanent theory	[43]
MSQRTA	U346	1024	0	0	0	1024	0	0	0	0	matrix problem	[42]
NLMSURF	U361	900	124	0	0	961	0	0	0	0	minimum surface	[56]
NONDIA	U379	10000	0	0	0	10000	0	0	0	0		[53]
NONSCOMP	U386	0	0	10000	1	9999	0	0	0	0		[40]
OBSTCLAL	U407	0	496	15129	0	15129	0	0	0	0	obstacle problem	[1]
ORTHREGD	C224	10003	0	0	0	10000	0	5000	0	0	orthogonal regression	[36]
PENALTY1	U451	1000	0	0	1000	1	0	0	0	0		[32]
POWELLSG	U484	10000	0	0	0	10000	0	0	0	0	extended Powel singular	[56]
READING2	C322	100	2	201	1	0	200	0	0	0	tide modelling	
SEMICON2	U509	0	2	1000	0	0	0	1000	0	0	physics	[1]
SINQUAD	U516	10000	0	0	0	10000	0	0	0	0	sines and quadratics	
SPMSQRT	U524	10000	0	0	0	16664	0	0	0	0	matrix problem	[42]
STEENBRA	C236	0	0	432	0	1	108	0	0	0	traffic equilibrium	[54]
SVANBERG	C258	0	0	1000	0	1000	0	0	0	1000	engineering	[55]
TOINTGSS	U559	10000	0	0	0	9998	0	0	0	0	Gaussian problem	[56]
TORSION4	U587	0	484	14400	1	29282	0	0	0	0	engineering	[1]
TQUARTIC	U539	10000	0	0	0	10000	0	0	0	0	arrowhead quartic	
TRIDIA	U548	10000	0	0	10000	0	0	0	0	0	tridiagonal quadratic	[53]
VAREIGVL	U610	5000	0	0	0	5000	0	0	0	0	numerical analysis	[1]
WOODS	U616	10000	0	0	10000	5000	0	0	0	0	extended Woods	[56]

Table 2: The test problem subset

6.2 The testing procedure

Before detailing the testing procedure, we recall the **default** algorithmic choice for LANCELOT:

- no variable/constraint scaling,
- a conjugate gradient linear solver is used with a banded preconditioner of semi-bandwidth 5 (band(5)),
- analytical second derivatives are used, as well as analytical gradients,
- an exact Cauchy point calculation is used,
- the ℓ_∞-norm is used for defining the trust region.

For our tests we also set the maximum number of iterations to 1000, the maximum cpu-time to 18000 seconds, the initial trust region radius to 1.0 and disabled all printing. The accuracy requirements were set to the LANCELOT defaults, that is $\epsilon_l = \epsilon_c = 10^{-5}$. We also turned the derivative checker on but chose to ignore its warning messages. This last choice is to reflect what would be a reasonable strategy for a cautious user. Of course, all derivatives were checked before the actual tests. For the sake of completeness, the default LANCELOT specification file is given in Figure 5.

```
BEGIN
   check-derivatives
   ignore-derivative-bugs
   exact-second-derivatives-used
   bandsolver-preconditioned-cg-solver-used 5
   exact-cauchy-point-required
   trust-region-radius 1.0D+0
   maximum-number-of-iterations 1000
   print-level -1
   start-printing-at-iteration 0
   stop-printing-at-iteration 1000
END
```

Figure 5: The LANCELOT default specification file

We next considered all the basic variants of this default choice, that is a choice of algorithmic options that differs in just one instance from the default. The basic variants are

scaling: automatic variable/constraint scaling is used, with scalings computed at the starting point (see Section 5.1),

mltf: a multifrontal direct linear solver is used (see Section 5.2.1),

semltf: a modified multifrontal direct linear solver is used (see Section 5.2.1),

noprc: no preconditioner is used within the conjugate gradient solver, i.e. $\bar{P}^{(k,j)} = I$ in (5.7) (see Section 5.2.2),

diagonal: a diagonal preconditioner is used both for the Cauchy direction and the conjugate gradient solver, together with a trust region scaling (see Section 5.2.2),

band(0): a diagonal preconditioner is used for the conjugate gradient solver (see Section 5.2.2),

band(1): a tridiagonal preconditioner is used for the conjugate gradient solver (see Section 5.2.2),

band(10): a 21-diagonals preconditioner is used for the conjugate gradient solver (see Section 5.2.2),

expband: an expanding band preconditioner is used for the conjugate gradient solver (see Section 5.2.2),

seprc: a full matrix preconditioner using the Schnabel-Eskow modified factorization is used for the conjugate gradient solver (see Section 5.2.2),

gmpsprc: a full matrix preconditioner using the Gill-Murray-Ponceléon-Saunders modified factorization is used for the conjugate gradient solver (see Section 5.2.2),

munksg: an incomplete factorization preconditioner is used for the conjugate gradient solver (see Section 5.2.2),

bfgs: the Broyden-Fletcher-Goldfarb-Shanno quasi-Newton formula is used to approximate second derivatives (see Section 5.3.5),

dfp: the Davidon-Fletcher-Powell quasi-Newton formula is used to approximate second derivatives (see Section 5.3.5),

psb: the Powell-Symmetric-Broyden quasi-Newton formula is used to approximate second derivatives (see Section 5.3.5),

sr1: the Symmetric Rank One quasi-Newton formula is used to approximate second derivatives (see Section 5.3.5),

appGCP: an approximate Cauchy point calculation is used (see Sections 3.2 and 5.4.1),

l2norm: the trust region is defined using the ℓ_2-norm, (see Section 5.4.2),

accBQP: an accurate solution to the BQP is sought (see Section 5.4.3).

To this list we added the **fdg** variant which uses finite difference approximation to gradients and the Symmetric Rank One quasi-Newton formula for approximating second derivatives (see Section 5.3.5). These variants and the default gives a list of 21 different algorithmic choices.

Note that the variants **scaling**, **mltf**, **semltf**, **expband**, **seprc**, **gmpsprc** and **munksg** depend on code from the Harwell Subroutine Library. Their use is therefore only possible for users with a suitable licence. As a consequence, they could not be selected as default for the package.

We then tested all of these 21 choices on the complete problem set, which amounted to running $21 \times 943 = 19803$ test cases. These tests were performed on two Digital DECstations 5000/200 with 48 MBytes of memory, using the Ultrix f77 compiler (version 3.0-2) without optimization[4]. The cpu-times on both machines were checked for consistency.

7 The numerical tests: results and discussion

It is of course impossible to detail the complete set of results obtained on nearly twenty thousands test cases. We will therefore present and discuss summaries and averages extracted from these results. A technical report containing the complete results is however available [16].

7.1 Reliability

We first present results on the reliability and failures on the 21 algorithmic variants. Results are given in Table 3, where the occurrences of the LANCELOT exit conditions are reported for all 21 variants in the case of the complete test set and the selected subset. The column headings correspond to the following possible situations.

succ: The minimization was successfully terminated.

infs: The package could not find a feasible point for the considered problem.

stall: The minimization could not progress further, the stepsize being smaller than relative machine precision. Not all runs terminated in this way are unsuccessful from the user's point of view, as it happens in several cases that the algorithm is "stalled" very near the solution.

mem: The workspace requires for handling the considered problem is larger than three millions double precision and/or three millions integer numbers.

iters: The run was terminated after 1000 iterations without convergence.

cpu: The run was terminated after 18000 cpu seconds without convergence.

error: An arithmetic error occurred in the subprograms evaluating the problem dependent functions and/or derivatives. This typically occurs when the iterates produced by the algorithm "wander off" the part of the feasible region where the objective and constraints are of manageable size.

From this table, we can draw the following conclusions.

[4] An error in the Fortran optimizer of this version prevented its use with the package.

Variant	Complete set (943 problems)							Selected subset (45 problems)						
	succ	stall	infs	mem	iters	cpu	error	succ	stall	infs	mem	iters	cpu	error
default	873	11	2	0	31	24	2	41	1	0	0	0	3	0
scaling	807	45	25	0	28	29	9	33	3	2	0	2	5	0
mltf	777	5	11	12	106	29	3	28	1	0	4	7	5	0
semltf	807	5	11	2	65	48	5	24	0	1	1	6	13	0
noprc	850	6	13	0	35	36	3	35	0	0	0	1	9	0
diagonal	744	1	16	0	153	23	6	31	0	1	0	7	6	0
band(0)	851	19	8	0	30	33	2	36	3	0	0	0	6	0
band(1)	859	18	4	0	34	25	3	39	0	0	0	0	6	0
band(10)	871	10	8	0	27	25	2	41	1	0	0	0	3	0
expband	864	7	8	3	25	26	10	33	0	0	2	2	8	0
seprc	878	11	7	2	22	21	2	38	0	0	1	0	6	0
gmpsprc	860	10	7	9	26	21	10	35	1	0	2	1	4	2
munksg	851	7	13	2	28	39	3	34	0	1	1	1	8	0
bfgs	786	15	12	0	88	26	16	32	3	1	0	6	3	0
dfp	621	57	15	0	203	40	7	23	7	1	0	10	4	0
psb	857	13	3	0	37	28	5	39	2	0	0	1	3	0
sr1	871	13	5	0	26	26	2	39	2	0	0	1	3	0
appGCP	841	23	10	0	29	29	11	37	1	1	0	0	5	1
l2norm	851	15	10	0	38	28	1	38	2	0	0	1	4	0
accBQP	863	11	4	0	16	48	1	36	0	0	0	0	9	0
fdg	784	19	11	0	88	28	13	32	2	0	0	7	4	0

Table 3: Successes and failures per variant

1. The reliability of the default algorithmic choice is good (92.5% on the complete problem set), and is only marginally surpassed by that of the Schnabel-Eskow preconditioner used in conjunction with conjugate gradients (93.1% on the complete set). The default variant has the best reliability on the selected subset (together with band(10)). We note that the reliability is globally lower for the subset; this is expected because the subset contains some of the most difficult problems.

   The default choice of a semi-bandwidth of 5 also seems to maximize reliability amongst the banded preconditioners, both for the complete problem set and the subset.

2. The SR1 update is the most reliable of the quasi-Newton methods tested, followed by the PSB method and, at some distance, by the BFGS method. The DFP shows the worst reliability. The perhaps surprisingly good performance of the PSB method could be partly explained by the observation that, although some problems in the test set are very badly scaled, this is not the case for the majority. This seems to be confirmed by the very

acceptable reliability score obtained by the conjugate gradient linear solver without any preconditioning. On the other hand, this remark does not explain the relatively poor performance of BFGS and DFP. The poorer performance of the BFGS and DFP updates is also partly explained by the fact that these updates must be skipped whenever they would result in a non positive definite element Hessian approximation.

The results on the problem subset confirm the above, where the unreliability of the DFP method reaches a rather unacceptable level.

3. The robustness of the best partitioned quasi-Newton scheme (SR1) appears to be excellent compared with the use of exact second derivatives, even for large problems. This approach therefore confirms its potential amongst quasi-Newton techniques for large-scale applications, at least from the reliability point of view.

4. The scaling variant does not show a globally improved robustness compared with the default. This illustrates the difficulty designing good automatic scaling procedures. It is however worthwhile to note that the scaling variant did solve badly scaled problems where other variants failed. Keeping such an option available therefore seems to be of some value, but it should not be used as a default.

5. The reliability of the SR1 and PSB methods are very comparable to that of the variants using exact second derivatives. The partitioned quasi-Newton approximation therefore seems quite effective when analytical Hessians are not available. This is also apparent in the subset.

6. The reliability of the diagonal variant is quite poor, compared with the other choices using exact second derivatives. In particular, it is significantly worse than that of the similar band(0) variant. Again, this observation is supported by the subset results.

7. It is somewhat surprising that the gmpsprc variant has a significantly lower reliability than the other full matrix preconditioner seprc on the complete test set. On the other hand the reliability of both variants is equivalent on the problem subset.

   For the complete set, the Gill-Murray-Ponceléon-Saunders technique seems to generate more arithmetic errors and to run out of memory more often than the Schnabel-Eskow method.

   On closer analysis, the occurrence of overflow with the Gill-Murray-Ponceléon-Saunders modified factorization seems to be due to numerical difficulties for some singular or nearly singular matrices. The observed problems are probably caused by the low value of the threshold under which eigenvalues are perturbed to ensure positive definiteness of the preconditioning matrix. According to [30], this threshold is set to the machine precision. A posteriori experiments with the threshold raised to (machine precision)$^{3/4}$ (as is used in the Schnabel-Eskow modification) however indicates that the overflow problems can be avoided. These observations are coherent with the conclusions of Schlick in [51], where she observes that enforcing a small modification $\bar{E}^{(k,j)}$ in (5.5) might not be beneficial for fast convergence.

The difference in memory requirements for the two methods is due to a possibly larger fill-in in the Gill-Murray-Poncelèon-Saunders technique caused by changes in the pivoting order to preserve stability. As the Schnabel-Eskow modified factorization maintains positive definiteness of the matrix during the factorization, no such changes are necessary. This observation is supported by the results for the direct analogs of these methods, namely semltf (corresponding to seprc) and mltf (using MA27 as gmpsprc).

The slightly higher number of memory requirement failures for mltf compared with gmpsprc is explained by the fact that the first strategy uses additional storage in order to retrieve directions of negative curvature from the matrix factors when suitable (see [9] for further details).

8. We also note the substantial gain in robustness obtained by using the full matrix factorizations as preconditioners. The variants seprc and gmpsprc are indeed significantly more reliable than their direct counterparts semltf and mltf.

9. The accBQP variant, being more computationally intensive, runs out of time most often. If we assume that some of the truncated computations would effectively terminate successfully, given additional time, this variant probably ranks as the most reliable, but at the expense of substantial additional effort.

10. The approximate calculation of the Generalized Cauchy point (the appGCP variant) appears to decrease the reliability compared with the default choice of an exact computation, although not dramatically. This is explained by the less conservative nature of this first strategy which tends to allow larger steps. These steps might then lead the iterates along different paths to the solution, sometimes across regions where the problem function evaluations cause numerical overflow, as is show by the larger number of error returns for the appGCP variant.

11. There does not seem to be a real robustness advantage in using an incomplete factorization preconditioner (munksg) over a banded one for the problems of our test set. One must however notice that discretized continuous problems do not constitute a majority of the tested cases. As incomplete factorizations have earned their good reputation on such problems, one could probably expect a better performance of the munksg variant if the proportion of discretized problems increased.

12. The use of the ℓ_2-norm in the trust region definition marginally deteriorates the robustness of the minimization, as can be checked by comparing the results obtained by the default and l2norm variants. This could well be due to the fact that more work is invested in the Generalized Cauchy Point calculation for the default variant, as explained in Section 5.4.2.

13. Using finite difference approximations for the first derivatives of the problem's function somewhat reduces the reliability of the package, but **fdg** still managed to solve 83% of the problems, a quite acceptable score.

We conclude our reliability analysis by noting that 919 of the 943 problems were solved by at least one variant, while 441 were solved by all of them. This indicates an excellent reliability of the complete package (97.5%) on our large test problem collection, but also the relative lack of robustness for certain algorithmic variants. This last observation is strengthened by examining the problem subset, where only 12 problems were successfully solved by all variants.

7.2 Number of minor iterations

We now start comparing the algorithmic variants for efficiency. To be fair, only two sets of runs can be compared for each variants: the runs that successfully produce a well specified critical points and the unsuccessful ones. We therefore remove from our comparison all runs for which the variant under consideration converged to a point whose associated optimal value does not correspond (within 0.001%) to the best critical value found for the problem. In total, 434 problems from the complete set and 8 from the subset were successfully solved (according to this criterion) by all methods. We shall confine our attention to these problems. Figure 6 indicates how many problems per variant were discarded.

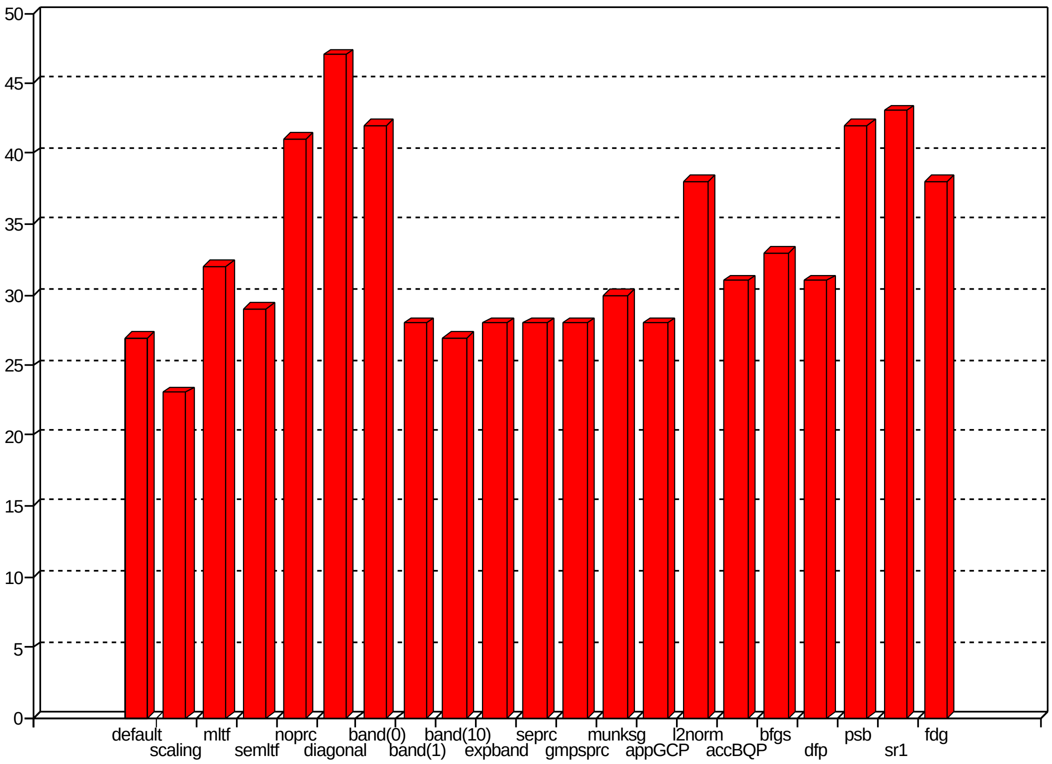

Figure 6: Number of runs to alternative critical points per variant

The figure reveals that convergence to an alternative critical point is obtained most often for the **diagonal** variant, although not very significantly.

We now turn our attention to the number of minor iterations required by the variants to find the solution. We recall that the problems objective function and constraints (if any) are evaluated exactly once per such iteration. Hence the number of minor iterations is equal to the numbers of functions evaluations.

Figure 7 shows the average number of iterations required for solution. Figure 8 presents an overall view of the relative ranking of the variants based on the number of iterations. To construct this figure, we considered all problems that were successfully solved by at least one variant. All 21 variants were ranked (where best means ranked number 1) for each of these 826 problems. We then counted the number of times that a given variant had a given rank. We finally clustered the obtained rankings in classes (ranks 1 to 3, 4 to 6, ...) which could be effectively displayed in a bar chart. For instance, the darker area in the bar corresponding to the seprc variant indicates that this variant is amongst the three best for 527 problems, an excellent performance. The interpretation of the total height of the bar is slightly different from the reliability scores presented in Table 3: successful runs producing a critical point with an optimal value different from the best one found (see Figure 6) are not accounted for.

Figure 9 presents the corresponding rankings for the selected subset of test problems.

We now draw some conclusions from these figures.

1. We immediately note the good results obtained by the semltf variant for the complete problem set. Although less reliable than its preconditioning counterpart seprc, it seems to require fewer iterations to converge when it does so, but the difference is admittedly marginal.

2. The full matrix preconditioner variants seprc and gmpsprc also show superior efficiency.

3. An excellent result is also observed for the accBQP variant. This is not a surprise. Indeed this variant puts more work in an iteration and one therefore expects that less of these more costly iterations are needed. This trend is however less marked on the larger problems of the subset.

4. The default variant appears to be reasonably efficient in terms of minor iterations, although not amongst the best. It is however remarkable that it is the variant whose behaviour is least often amongst the worst ranking, as is shown by the size of the class corresponding to the lowest rankings (in Figure 8). This last characteristic is displayed by the seprc variant on the subset.

5. The diagonal variant again shows poor behaviour. We noted above that it is comparatively unreliable, but it also seems to be quite inefficient even when it manages to converge to the problem's solution. The same comment applies, to a lesser extent, to the dfp variant on the complete set.

6. The l2norm variant has a relatively poor ranking. A possible reason for this behaviour is the fact that minor iterations are stopped earlier (see Section 5.4.2) than for other variants.

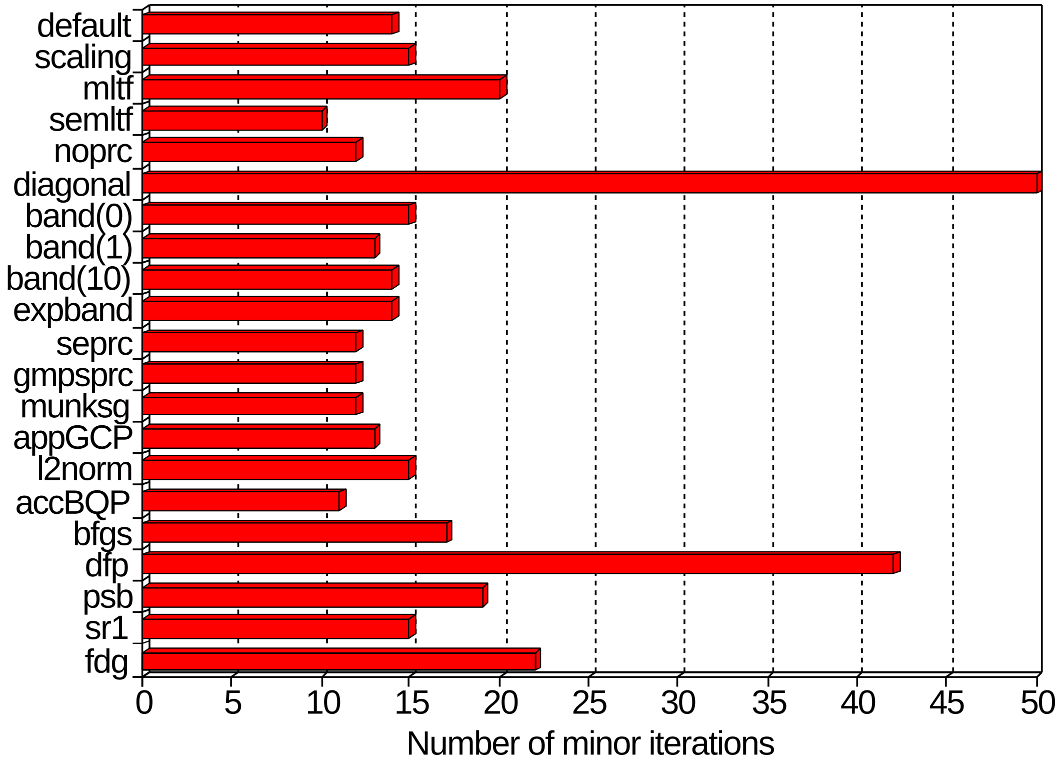

Figure 7: Average number of iterations for 434 problems solved by all variants

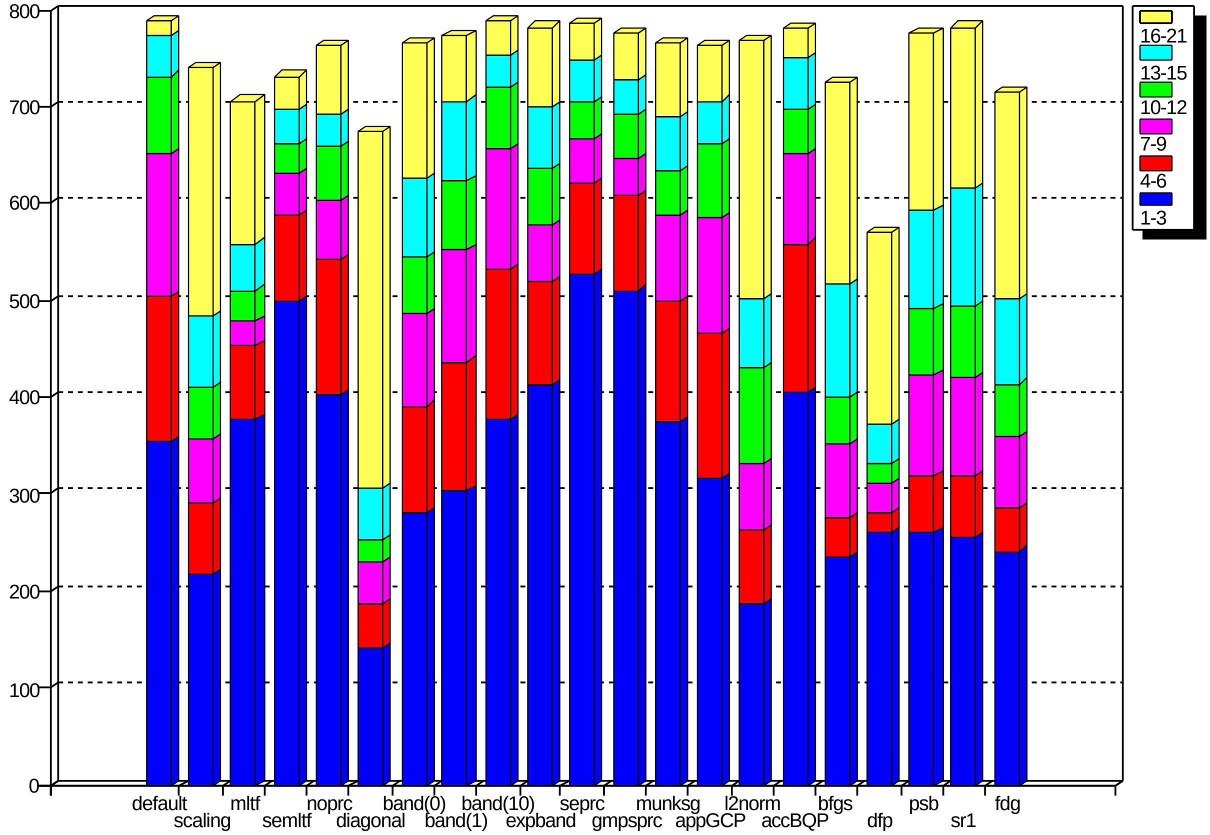

Figure 8: Ranking per iterations for 826 problems solved by at least one variant

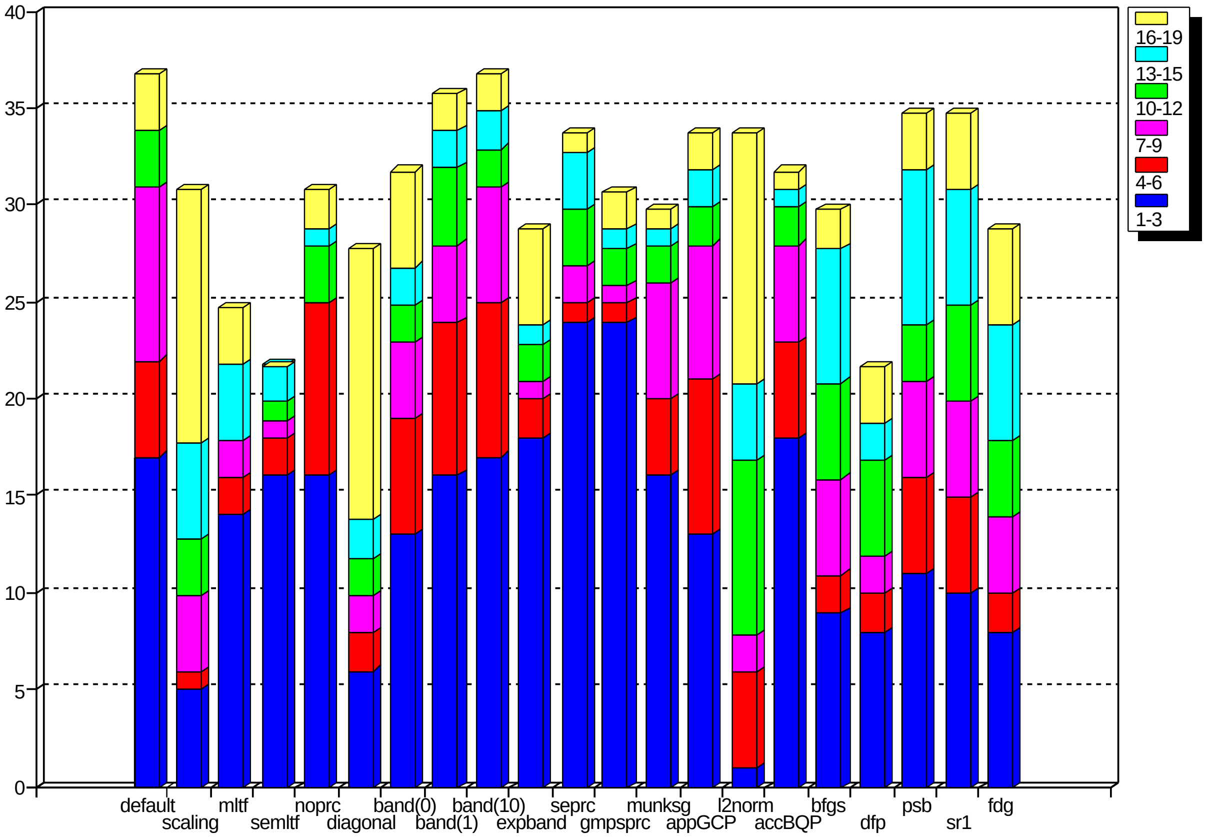

Figure 9: Ranking per iterations for 40 problems of the subset solved by at least one variant

7.3 Number of cg-iterations

After the number of minor iterations, we now examine the total number of conjugate gradient iterations per minor iteration required to solve the test problems by each variant using an iterative linear solver. What is really compared in this section is the overall effect of the various preconditioners and, to some extent, the conditioning of the Hessian matrices generated by the different variants.

Figure 10 shows the average number of cg-iterations per minor iteration and per problem variable, the average being taken on the 441 problems in the complete set that were successfully solved by all methods. This measure indicates how many cg-iterations were performed on average, compared to the problem size. Since conjugate gradients are expected to terminate in at most n cg-iterations on a system of size n, the reported measure are all between zero and one. We note that the measure is approximate for two reasons. Firstly, the number of free variables at any given iterations can be lower than the number of problem variables. Secondly, the conjugate gradient iterations may have to be restarted when bounds are encountered. We however believe the comparison amongst variants to be instructive. Figure 11 presents the same measure taken on the 8 problems of the subset that were successfully solved by all variants.

1. As anticipated, the full-matrix preconditioners are the clear winners in terms of number of cg-iterations. This behaviour is even more marked on the problem subset.

2. Another expected conclusion is that the quality of the preconditioner seems to increase with the semibandwidth, when a band preconditioner is used. This is clearly apparent when examining the results for the complete set for noprc, band(0), band(1), default (which is equivalent to band(5)), band(10) and expband. For the larger problems of the subset, one can however observe a superior behaviour of the small bandwith variants, but this might be due to the structure of the 8 considered problems.

3. The diagonal preconditioning uses less cg-iterations per minor steps than noprc, as shown by the figure, but this is partly caused by the very large number of these minor iterations, where only a few cg steps are necessary. Indeed the absolute number of cg-iterations for this variant exceeds that of noprc.

4. The incomplete factorization preconditioner munksg shows superior behaviour. Indeed its performance is comparable to that of the full-matrix variants.

5. Solving the BQP accurately of course requires more cg-iterations, and we observe this effect when comparing the default and accBQP.

6. Why the variant appGCP requires more cg-iterations than the default is not clear. One possible explanation would be that the average model reduction achieved at the GCP is smaller when the approximate strategy is used to compute this point than with the exact technique, subsequently requiring additional work for the model minimization in the next stage.

7. The scaled variant **scaling** is somewhat less efficient than the **default** unscaled variant on the complete problem set, which is again an indication that scaling should not be applied blindly to every problem. Its performance is however improved on the larger problems of the subset.

8. The quasi-Newton approximations to the second derivatives does not seem to generate matrices that are, on average, worse conditioned than their analytic values, as is shown by the comparable level for the **default**, **bfgs**, **dfp**, **psb** and **sr1** variants. The fact that gradients are estimated by differences in **fdg** does not seem to impact the conditioning of the Hessian either, as can be seen by comparing this variant with **sr1**.

9. The reported measures are typically smaller for the subset than for the complete problem set. This is anticipated as conjugate gradient solvers often require a number of iterations that is more dependent on conditioning and eigenvalue distribution than on system size. Increasing size therefore produce lower measures if one assume that the larger problems have an eigenvalue structure that is, on average, not worse than that of smaller ones.

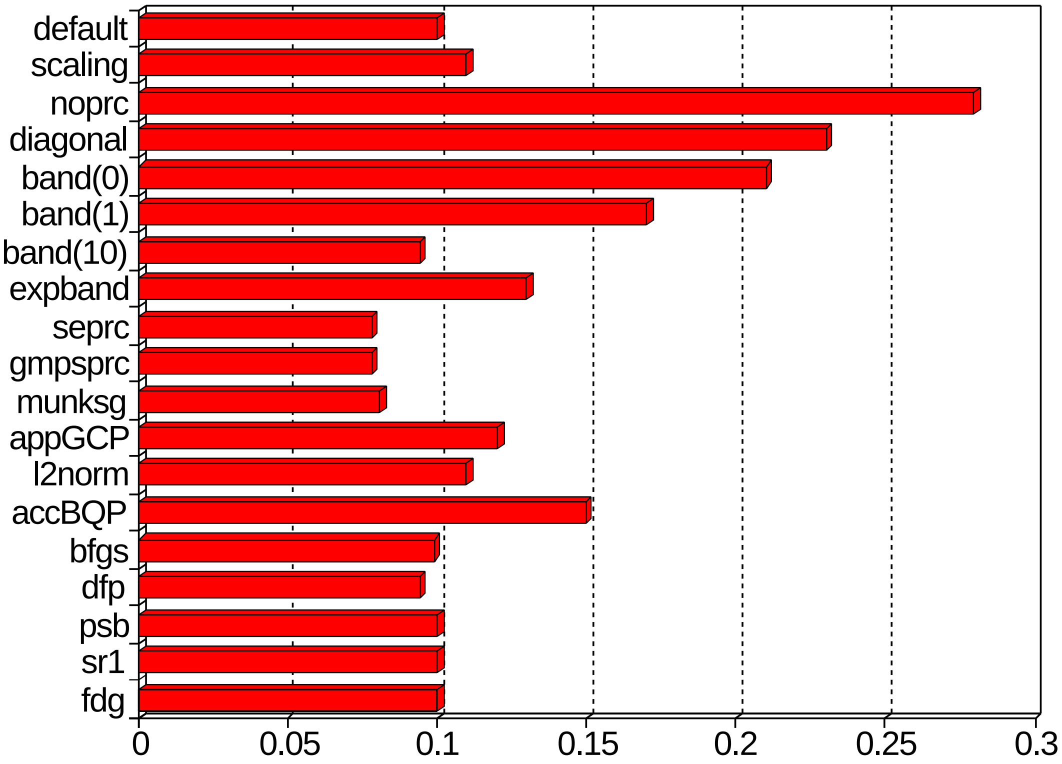

Figure 10: Average number of cg-iterations per minor iteration for 434 problems solved by all variants

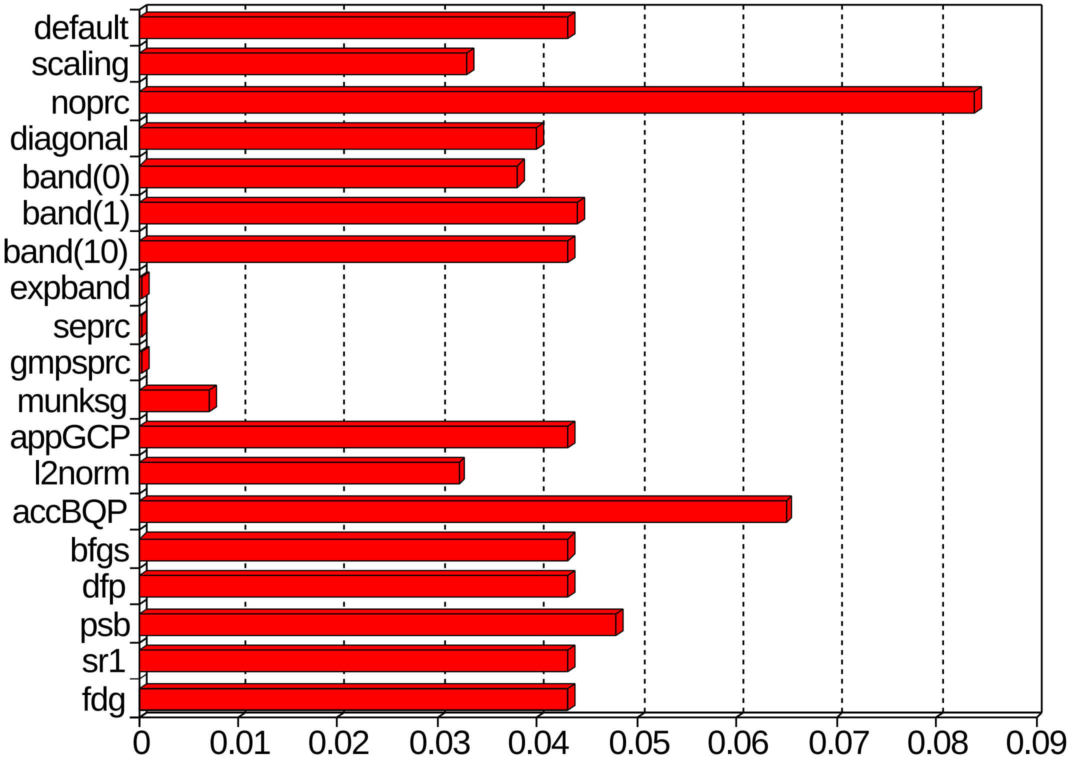

Figure 11: Average number of cg-iterations per minor iteration for 8 problems solved by all variants

7.4 Computational effort

We next compare our 21 algorithmic variants on the basis of their requirements in cpu-time.

Figure 12 shows the average cpu-time (in seconds) required for solution, the average being taken on the 434 problems in the complete set that were successfully solved by all methods. Figure 13 presents a overall view of the relative ranking of the variants based on cpu-time. This figure was constructed in the same way as Figure 8. Figure 14 presents the corresponding ranking results for the selected subset of test problems.

Some interesting conclusions can be drawn from these figures.

1. The incomplete factorization preconditioner **munksg** appears to provide the best average performance in terms of computational effort. However, its ranking relative to the other variants is good but not amongst the best. This means that it is mostly efficient on some of the harder or larger problems. This observation is reinforced by the detail of the results obtained for this variant on the problems of the selected subset: markedly leading on average but best on few problems.

2. The results obtained by the **semltf** variant are very interesting. Although its ranking compared with the other variant is amongst the best, its average performance is the poorest. This is caused by the poor behaviour of the variant on a few large unconstrained problems where the Hessian matrix is indefinite in the early iterations. In these cases, the strategy to move along a direction of negative curvature, as in the iterative variants and in **mltf**, seems more appropriate than repeatedly calculating a modified Newton direction in smaller and smaller subspaces (corresponding to faces of the trust region), each time recomputing a suitably modified factorization. It should however be noted that, despite its strong effect on average scores, this behaviour occurs rarely, as can be seen from the comparative ranking of the variant.

3. The full matrix preconditioned variants **seprc** and **gmpsprc** appear to be quite efficient on average. We note the good relative ranking of **gmpsprc** for the problem subset, where it seems to be very efficient for some problems, while spending much effort in others. This behaviour is not surprising because the efficiency of full matrix preconditioners (and direct solvers) heavily depends on the amount of fill-in during the factorization, a highly problem dependent feature.

4. The global efficiency of **mltf** is only marginally worse than that of its iterative counterpart **gmpsprc**. As was observed in [9], the behaviour of this direct method is best on convex problems with relatively little fill-in.

5. Quite interestingly, the **scaling** variant does not seem to be handicapped by the additional work required by computing and handling the variable and constraints "typical" values. It is indeed quite comparable to the **default** option.

6. The relatively acceptable performance of the **noprc** variant seems to confirm the remark that most of the test problems are reasonably well scaled.

7. The behaviour of banded preconditioners with varying semi bandwidth is worth a comment. We first note the good performance of the tridiagonal preconditioner (band(1)), both on the complete problem set and on the subset. The band(10) variant uses more cpu-time as the advantage of improved preconditioning is offset by the higher price of the preconditioner, an effect already noticeable with the satisfying results obtained with the default band(5) variant. The good performance of the expanding band variant expband, compared with band(10), seems to be due to the general sparse storage scheme used, which is preferable to the band storage for matrices with higher bandwidth.

8. The more costly iterations of accBQP clearly cause the relatively large average cpu-time of this variant on the complete problem set. However, as the expense of cpu-time is mostly confined to large problems, and as there are comparatively few such problems in the complete test set, the method ranks reasonably highly. This observation is strengthened by the poor ranking of this variant for the large problems of the subset, where it never ranks amongst the three best.

9. The appGCP variant is markedly less efficient than default on the complete test set, although its relative ranking is better.

10. The diagonal variant pays in cpu-time the price for its many minor iterations, giving overall rather poor performance, both on the complete problem set and on the subset.

11. Amongst the quasi-Newton variants, sr1 appears to be the most efficient, followed in order by bfgs, dfp and psb.

12. The work involved in approximating the gradients by differences causes fdg to be slower than sr1 on average, but not by a large margin. But this effect is enough to cause the relative ranking of fdg to fall substantially behind that of sr1.

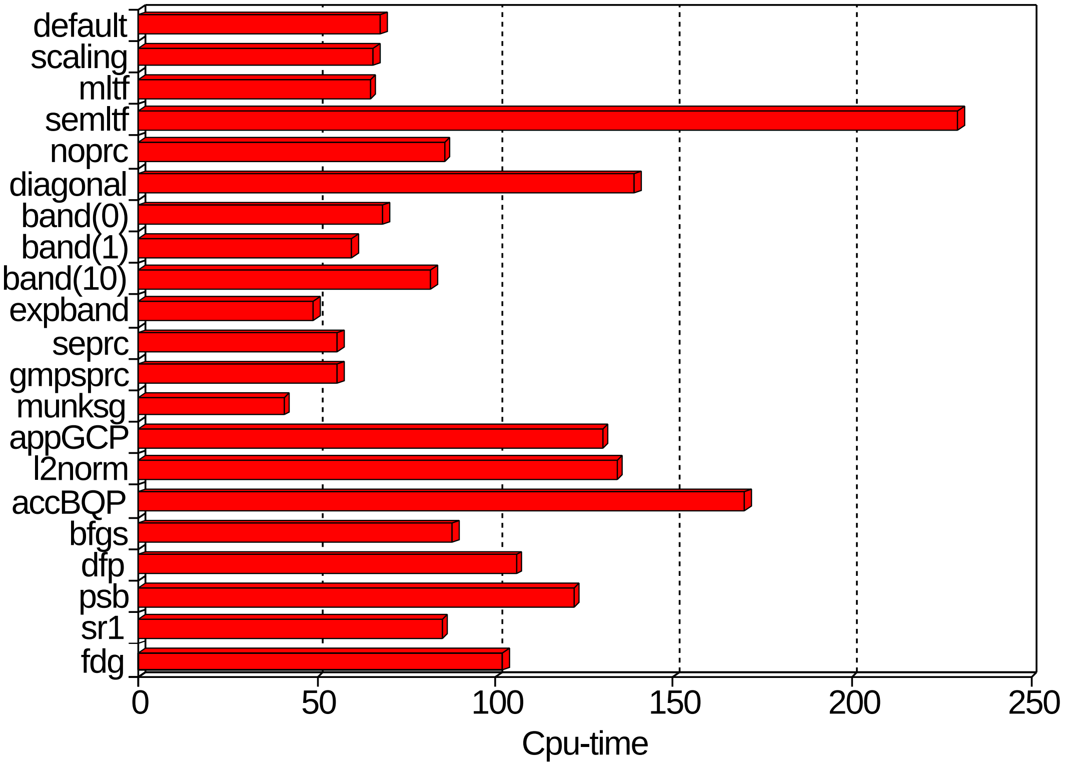

Figure 12: Average cpu-time for 441 problems solved by all variants

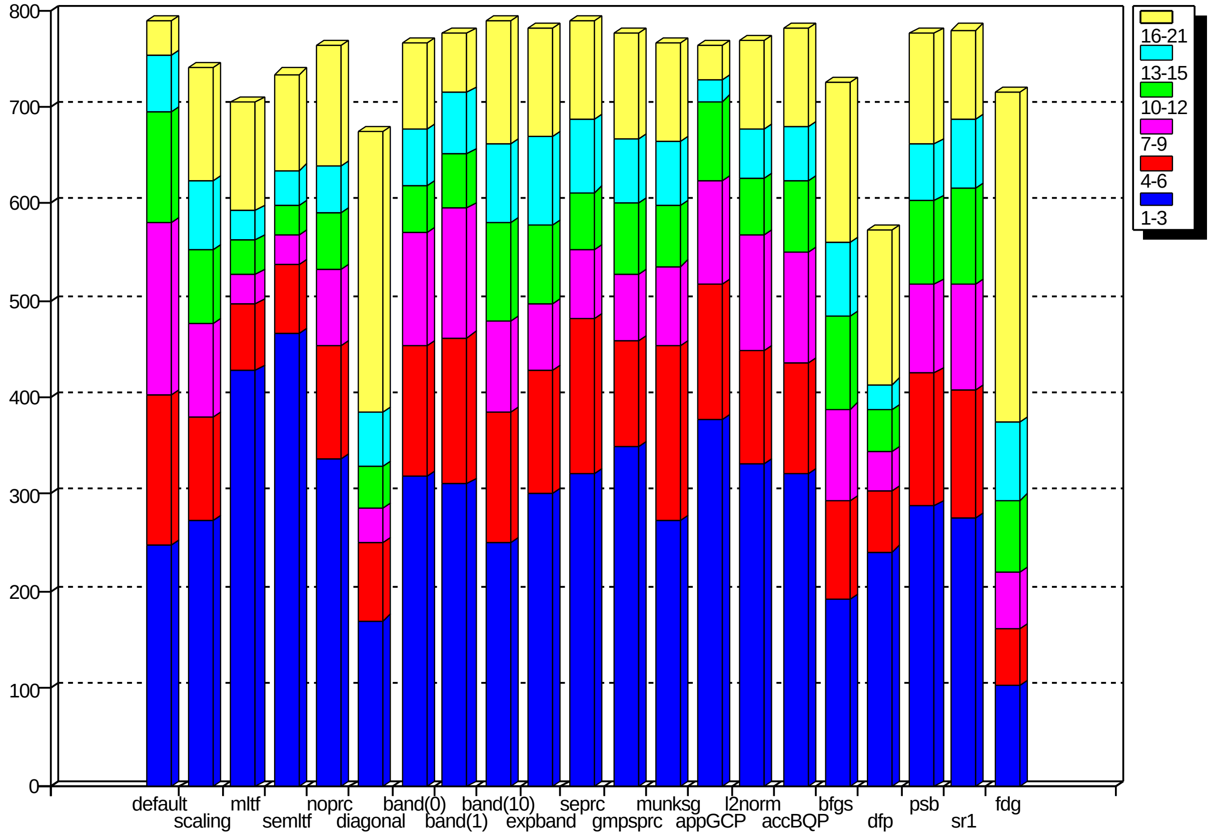

Figure 13: Ranking per cpu-time for 826 problems solved by at least one variant

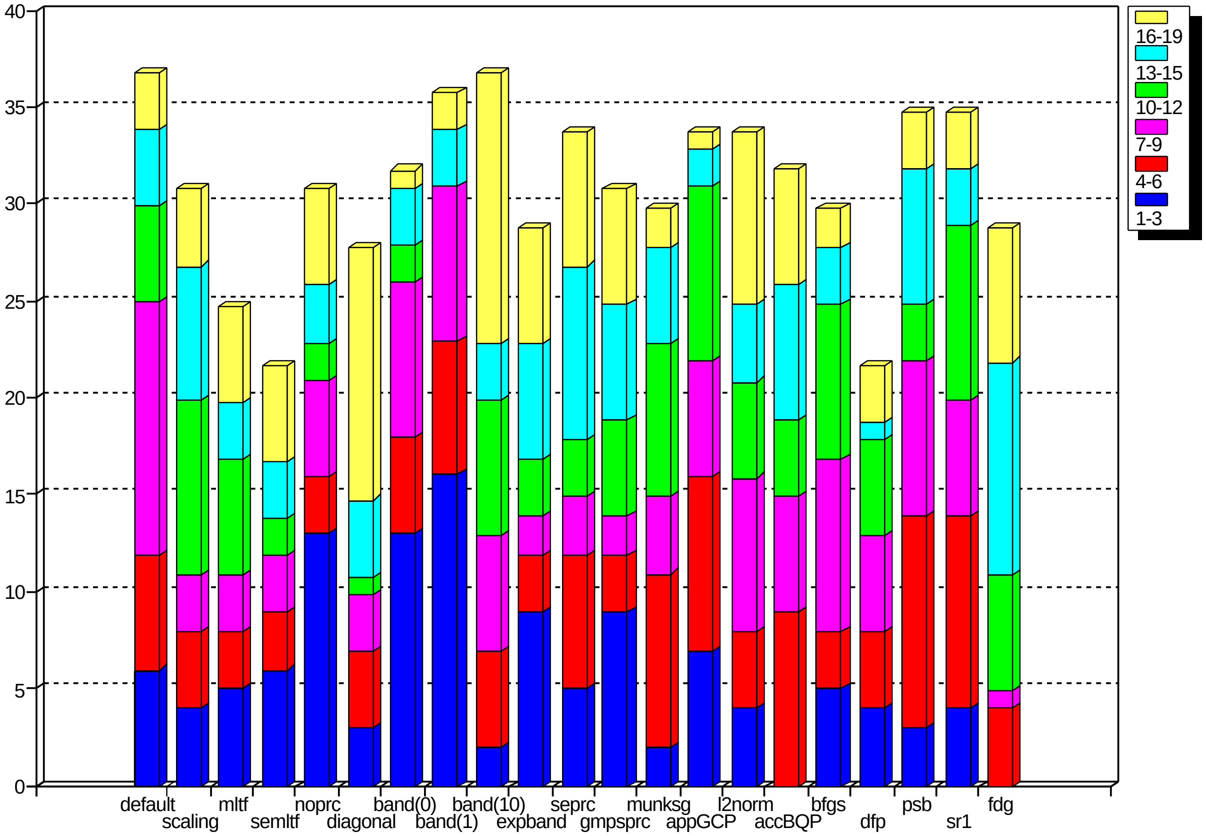

Figure 14: Ranking per cpu-time for 40 problems of the subset solved by at least one variant

7.5 Additional comments

We did not discuss above the relative number of unsuccessful iterations for each variant. This number is *on average* below one per problem for each variant, except for the dfp variant for which an average of six such iterations per problem are observed. This again confirms the poor performance of this last variant. It also seems to indicate that the trust region management used in LANCELOT is adequate for handling a large class of nonlinear problems.

Besides its algorithmic choices, LANCELOT allows the user to select a number of non algorithmic options, such as element and group derivative checking, level of printout and frequency at which intermediate data is saved for a possible subsequent restart. None of these options has a significant impact on the overall execution time of the package. The only observable increase in cpu-time occurs when a very detailed printout is required at every iteration of a large scale problem. As one would expect, this effect is slightly more marked for constrained cases, where the details of the major iterations have to be printed as well.

We finally indicate some weak points of LANCELOT (Release A) that we have observed in examining the detailed runs, but that cannot be inferred directly from the summaries presented above.

1. When the number of inequality constraints is large compared with the number of variables, the package currently adds slack variables to transform all inequalities into equalities, which results in a substantial increase in the effective problem size. Although convergence is usually obtained, the computational effort can be relatively large compared with method that use inequality constraints as such (see [15], for instance). The authors are well aware of this aspect of their implementation, and have recently given in [14] a method to overcome this difficulty.

2. No special provision is made in the present code for linear network constraints, or even for linear constraints. Again, LANCELOT seems to be robust in that convergence is obtained for problems with this kind of structure, but special purpose algorithms are often much more efficient.

3. The ability of the generalized Cauchy point to determine the correct active set is disappointing in practice. In many examples, the correct active set is actually found in the conjugate gradient or direct matrix improvement beyond the GCP, at considerable cost. Although the GCP asymptotically identifies the correct active set as predicted by the theory (see [10], for instance), this is often at the end of a long calculation. A stategy treating the bounds through barrier functions (as proposed in [15]) would therefore appear to be a useful alternative.

8 Conclusions

We first described the algorithms contained in Release A of the LANCELOT package for large-scale nonlinear optimization in detail. We also analyzed the user selectable variants at this level.

We finally presented and discussed the results of extensive numerical tests with these variants.

The main conclusions of these tests, as far as the package is concerned, are as follows.

1. The package is capable of solving a wide class of nonlinear optimization problems, including many large-scale examples.
2. The package is especially efficient for unconstrained and bound constrained problems and for generally constrained problems for which the number of constraints does not substantially exceeds the problem dimension.
3. The default algorithmic choice in the package appears to be reliable and acceptably efficient.
4. Some algorithmic choices (ℓ_2 trust region, diagonal rescaling, and DFP quasi-Newton updating) seem of limited utility and could probably be removed from future releases of the package.
5. Some other choices (automatic scaling, accurate solution of the inner BQP) should not be used automatically, but may provide excellent behaviour on some harder problems.

Beyond the tests of the LANCELOT package, our tests also reveal the following points of more general interest.

1. The difficulty of solving a problem is more often linked to its degree of nonlinearity than to its size.
2. The use of direct factorization appears to be most robust when used as preconditioners for a conjugate gradient linear solver.
3. The use of exact second derivatives is recommended whenever available. However, the partitioned Symmetric Rank One technique, as embedded in the package, gives very satisfactory reliability and efficiency when this is not the case.
4. When analytical derivatives are not available, finite difference approximations to the gradients coupled with SR1 quasi-Newton Hessian updating is an acceptably robust and efficient technique, even for large problems.
5. The use of full factorizations appears to be efficient and reliable for the class of problems analyzed in this paper. It is however expected that this technique would show stronger limitations if even larger problems were considered. In contrast, banded preconditioners, already satisfactory in the present study, would probably extend well to larger problems.

Of course, only continued experience with LANCELOT will really show its strengths and weaknesses. The authors very much hope to be informed by the users of the package of the (undoubtly many) aspects where improvements are possible. Progress is expected to come both from the point of view of the implemented algorithms (see [15] and [19] for possible directions) and from that of the implementation details themselves. At this stage, the results discussed above certainly offer the hope that the software will prove useful in the increasingly important arena of large-scale nonlinear optimization.

9 Acknowledgements

The authors are indebted to Michel Bierlaire and Didier Burton, whose help was invaluable in producing the graphics of this paper and in maintaining the workstation network during the 8 months of nearly uninterrupted computation required for obtaining the results presented here. Thanks are also due to Ingrid Bongartz and Peihuang Chen for detecting and correcting mistakes in our test problems. The authors are also grateful for the support provided for their travels across the Atlantic by NATO grant 890867.

References

[1] B. M. Averick, R. G. Carter, and J. J. Moré. The Minpack-2 test problem collection (preliminary version). Technical Report ANL/MCS-TM-150, Argonne National Laboratory, Argonne, USA, 1991.

[2] B. M. Averick and J. J. Moré. The Minpack-2 test problem collection. Technical Report ANL/MCS-TM-157, Argonne National Laboratory, Argonne, USA, 1991.

[3] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: a collection of constrained and unconstrained test examples for nonlinear programming. Technical Report (in preparation), Department of Mathematics, FUNDP, Namur, Belgium, 1992.

[4] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. A numerical comparison between the LANCELOT and MINOS packages for large-scale nonlinear optimization. Technical Report (in preparation), Department of Mathematics, FUNDP, Namur, Belgium, 1992.

[5] A. G. Buckley. Test functions for unconstrained minimization. Technical Report CS-3, Computing Science Division, Dalhousie University, Dalhousie, Canada, 1989.

[6] J. V. Burke and J. J. Moré. On the identification of active constraints. *SIAM Journal on Numerical Analysis*, 25(5):1197–1211, 1988.

[7] J. V. Burke, J. J. Moré, and G. Toraldo. Convergence properties of trust region methods for linear and convex constraints. *Mathematical Programming, Series A*, 47(3):305–336, 1990.

[8] P. H. Calamai and J. J. Moré. Projected gradient methods for linearly constrained problems. *Mathematical Programming*, 39:93–116, 1987.

[9] A. R. Conn, N. I. M. Gould, M. Lescrenier, and Ph. L. Toint. Performance of a multifrontal scheme for partially separable optimization. Technical Report 88/04, Department of Mathematics, FUNDP, Namur, Belgium, 1988.

[10] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM Journal on Numerical Analysis*, 25:433–460, 1988. See also same journal 26:764–767, 1989.

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

[12] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. An introduction to the structure of large scale nonlinear optimization problems and the LANCELOT project. In R. Glowinski and A. Lichnewsky, editors, *Computing Methods in Applied Sciences and Engineering*, pages 42–51, Philadelphia, USA, 1990. SIAM.

[13] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2):545–572, 1991.

[14] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Exploiting structure when using slack variables. LANCELOT B working note 1, Department of Mathematics, FUNDP, Namur, Belgium, 1992.

[15] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. A globally convergent Lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds. Technical Report 92/07, Department of Mathematics, FUNDP, Namur, Belgium, 1992.

[16] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Intensive numerical tests with LANCELOT (Release A): the complete results. Technical Report 92/15, Department of Mathematics, FUNDP, Namur, Belgium, 1992.

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

[18] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. On the number of inner iterations per outer iteration of a globally convergent algorithm for optimization with general nonlinear equality constraints and simple bounds. In D.F Griffiths and G.A. Watson, editors, *Proceedings of the 14th Biennal Numerical Analysis Conference Dundee 1991*. Longmans, 1992.

[19] A. R. Conn, Nick Gould, and Ph. L. Toint. Convergence properties of minimization algorithms for convex constraints using a structured trust region. Technical Report 92/11, Department of Mathematics, FUNDP, Namur, Belgium, 1992.

[20] A. R. Curtis and J. K. Reid. On the automatic scaling of matrices for Gaussian elimination. *Journal of the Institute of Mathematics and its Applications*, 10:118–124, 1972.

[21] R. S. Dembo. A primal truncated-newton algorithm with application to large-scale nonlinear network optimization. Technical Report 72, Yale School of Management, Yale University, New Haven, USA, 1984.

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

[23] L. C. W. Dixon and Z. Maany. A family of test problems with sparse Hessian for unconstrained optimization. Technical Report 206, Numerical Optimization Center, Hatfield Polytechnic, Hatfield, UK, 1988.

[24] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct methods for sparse matrices.* Clarendon Press, Oxford, UK, 1986.

[25] I. S. Duff and J. K. Reid. MA27: A set of Fortran subroutines for solving sparse symmetric sets of linear equations. Report R-10533, AERE Harwell Laboratory, Harwell, UK, 1982.

[26] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, 9(3):302–325, 1983.

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

[28] A. George and J. W.-H. Liu. *Computer solution of large sparse positive definite systems.* Prentice-Hall, Englewood Cliffs, USA, 1981.

[29] H. Gfrerer. Globally convergent decomposition methods for nonconvex optimization problems. *Computing*, 32:199–227, 1984.

[30] P. E. Gill, W. Murray, D. B. Ponceléon, and M. A. Saunders. Preconditioners for indefinite systems arising in optimization. SOL 90-8 Technical Report, Department of Operations Research, Stanford University, California, USA, 1990.

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

[32] P. E. Gill, W.Murray, and R. A. Pitfield. The implementation of two revised quasi-newton algorithms for unconstrained optimization. Technical Report NAC11, National Physical Laboratory, Teddington, UK, 1972.

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

[34] A. Griewank and Ph. L. Toint. On the unconstained optimization of partially separable functions. In M. J. D. Powell, editor, *Nonlinear Optimization 1981*, pages 301–312, London and New York, 1982. Academic Press.

[35] A. Griewank and Ph. L. Toint. Partitioned variable metric updates for large structured optimization problems. *Numerische Mathematik*, 39:429–448, 1982.

[36] M. Gulliksson. *Algorithms for Nonlinear Least Squares with Applications to Orthogonal Regression.* PhD thesis, Institute of Information Processing, University of Umeå, S-901 87 Umeå, Sweden, 1990.

[37] W. W. Hager. Multipliers methods for nonlinear optimal control. *SIAM Journal on Numerical Analysis*, 27(4):1061–1080, 1990.

[38] W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes.* Springer Verlag, Berlin, 1981. Lectures Notes in Economics and Mathematical Systems 187.

[39] K. M. Irani, M. P. Kamat, H. F. Walker, and L. T. Watson. Experiments with conjugate gradient algorithms for homotopy curve tracking. *SIAM Journal on Optimization*, 1(2):222–251, 1991.

[40] M. Lescrenier. *Towards the use of supercomputers for large scale nonlinear partially separable optimization.* PhD thesis, Department of Mathematics, FUNDP, Namur, Belgium, 1989.

[41] M. Lescrenier. Convergence of trust region algorithms for optimization with bounds when strict complementarity does not hold. *SIAM Journal on Numerical Analysis*, 28(2):476–495, 1991.

[42] D. C. Liu and J. Nocedal. Test results of two limited memory methods for large scale optimization. *Mathematical Programming, Series B*, 45(1):503–528, 1989.

[43] H. Minc. Theory of permanents 1982-1985. *Linear Algebra and Applications*, 21:109–148, 1987.

[44] J. J. Moré. Trust regions and projected gradients. In M. Iri and K. Yajima, editors, *System Modelling and Optimization*, volume 113, pages 1–13, Berlin, 1988. Springer Verlag. Lecture Notes in Control and Information Sciences.

[45] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, 1981.

[46] J. J. Moré and G. Toraldo. Algorithms for bound constrained quadratic programming problems. *SIAM Journal on Optimization*, 1(1):93–113, 1991.

[47] N. Munksgaard. Solving sparse symmetric systems of linear equations by preconditioned conjugate gradients. *ACM Transactions on Mathematical Software*, 6:206–219, 1980.

[48] B. A. Murtagh and M. A. Saunders. MINOS 5.1 USER'S GUIDE. Technical Report SOL83-20R, Department of Operations Research, Stanford University, Stanford, USA, 1987.

[49] J. Nocedal. Solving large nonlinear systems of equations arising in mechanics. In J. P. Hennart, editor, *Numerical Analysis: Proceedings, Cocoyoc, Mexico 81*, Berlin, 1982. Springer Verlag.

[50] K. Schittkowski. *More Test Examples for Nonlinear Programming Codes.* Springer Verlag, Berlin, 1987. Lecture notes in economics and mathematical systems, volume 282.

[51] T. Schlick. Modified Cholesky factorizations for sparse preconditioners. *SIAM Journal on Scientific and Statistical Computing*, (to appear), 1992.

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

[53] D. F. Shanno. On variable metric methods for sparse Hessians. *Mathematics of Computation*, 34:499–514, 1980.

[54] P. A. Steenbrink. *Optimization of transport networks.* J. Wiley and Sons, London, 1974.

[55] K. Svanberg. Method of moving asymptots – a new method for structural optimization. *Int. J. Num. Meth. Eng.*, 24:359–373, 1987.

[56] Ph. L. Toint. Test problems for partially separable optimization and results for the routine PSPMIN. Technical Report 83/4, Department of Mathematics, FUNDP, Namur, Belgium, 1983.

[57] Ph. L. Toint. Global convergence of a class of trust region methods for nonconvex minimization in Hilbert space. *IMA Journal of Numerical Analysis*, 8:231–252, 1988.

[58] Ph. L. Toint and D. Tuyttens. On large scale nonlinear network optimization. *Mathematical Programming, Series B*, 48(1):125–159, 1990.

[59] Ph. L. Toint and D. Tuyttens. LSNNO: a Fortran subroutine for solving large scale nonlinear network optimization problems. *ACM Transactions on Mathematical Software*, (to appear), 1992.