GALAHAD, A LIBRARY OF THREAD-SAFE FORTRAN 90 PACKAGES FOR LARGE-SCALE NONLINEAR OPTIMIZATION

by N.I.M. Gould[1], D. Orban[2] and Ph. L. Toint[3]

Report 02/8 4 May 2002

[1] Computational Science and Engineering Departement
Rutherford Appleton Laboratory,
Chilton, Oxfordshire, England
Email: gould@rl.ac.uk

[2] ECE Department, Northwestern University,
Evanston Il 60208, USA
Email: orban@ece.northwestern.edu

[3] Department of Mathematics, University of Namur,
61, rue de Bruxelles, B-5000 Namur, Belgium,
Email: philippe.toint@fundp.ac.be

GALAHAD, a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization

Nicholas I. M. Gould, Dominique Orban and Philippe L. Toint

4 May 2002

Abstract

In this paper, we describe the design of version 1.0 of GALAHAD, a library of Fortran 90 packages for large-scale large-scale nonlinear optimization. The library particularly addresses quadratic programming problems, containing both interior point and active set variants, as well as tools for preprocessing such problems prior to solution. It also contains an updated version of the venerable nonlinear programming package, LANCELOT.

1 Introduction

We released our large-scale nonlinear programming package LANCELOT (see Conn, Gould and Toint, 1992) late in 1991. Over the intervening years, LANCELOT A has been used by a large number of people, both via free source downloads from our WWW sites and by means of the NEOS facility (see Czyzyk, Mesnier and Moré, 1998) from the Optimization Technology Center at Argonne National Laboratory and Northwestern University in the USA. It is fair to say that we (and others) recognised its limitations right from the outset, and it has long been our goal eventually to provide a suitable successor.

LANCELOT A was written in Fortran 77. Despite the widespread availability of good Fortran 77 compilers, the limitations of the language, particularly the absence of standardised memory allocation facilities, proved to be a serious limitation, especially for large problems. Work started in 1995 to re-implement LANCELOT A in Fortran 90, particularly to take advantage of the new language's memory manipulation features and array constructs, and by 1997 we had a working prototype of the improved LANCELOT B. We had chosen to stay with Fortran rather than, say, C partially because many of the package's key (external) components, most especially the HSL (2002) sparse matrix codes produced by our colleagues, are all Fortran based, and also because we believed (and still believe) Fortran 90 capable of providing all of the facilities we needed.

Regrettably, at or around that time, a number of our colleagues had started to release the results of comparative tests of their new codes (for instance SNOPT, see Gill, Murray and Saunders, 2002, LOQO, see Vanderbei and Shanno, 1999, KNITRO, see Byrd, Hribar and Nocedal, 1999, and FilterSQP, see Fletcher and Leyffer, 2002) against LANCELOT A, and the results made frankly rather depressing reading for us (see Dolan and Moré, 2000, Benson, Shanno and Vanderbei, 2001, and Chin, 2001), LANCELOT often (but far from always) being significantly out performed. Quite clearly, the promise always held for large-scale SQP methods, based on how they outperformed other techniques in comparative tests such as those due to (Hock and Schittkowski 1981), was now being realised, and the limit of what might be achieved by augmented Lagrangian methods such as LANCELOT A had likely been reached.

Reluctantly, we abandoned any plans to release LANCELOT B at that time, and turned our attention instead to SQP methods. To our minds, there had never really been much doubt that SQP methods would be more successful in the long term, but there had been general concerns over how to (approximately) solve their all-important (large-scale) quadratic programming (QP) subproblems. Thus, we decided that our next goal should be to produce high-quality QP codes for eventual incorporation in our own SQP algorithm(s). Since we believe that there might be considerable interest from others in such codes, we have decided to release these before we have finalised our SQP solver(s). And since we realised that far from producing a single package, we are now in effect building a library of independent but inter-related packages, we have chosen to release an (evolving) large-scale nonlinear optimization library, GALAHAD.

In some sense GALAHAD Version 1.0 is a stop-gap, since, although it includes the upgraded B version of LANCELOT, we doubt seriously whether LANCELOT B is a state-of-the-art solver for general nonlinear programming problems. What GALAHAD V1.0 does provide are the quadratic programming solvers and related tools that we anticipate will allow us to develop the next-generation SQP solvers which we intend to introduce in Version 2 of the library.

2 Library contents

Each package in GALAHAD is written as a Fortran 90 module, and the codes are all threadsafe. The default precision for real arguments is double, but this is easily transformed to single in a UNIX environment using provided `sed` scripts. Each package has accompanying documentation and a test program. The latter attempts to execute as much of the package as is realistically possible—the fact that some of the packages are intended for nonlinear problems makes it difficult to ensure that every statement is executed, since some segments of code are intended to cope with pathological behaviour that cannot be ruled out in theory but which nevertheless seem never to occur in practice. Options to packages may be passed both directly, through subroutine arguments, and indirectly, via option-specification files. The second mechanism is particularly useful when there is a hierarchy of packages for which a user wishes to change an option for one of the dependent packages without recompilation.

2.1 Overview

GALAHAD comprises the following major packages:

`LANCELOT B` is a sequential augmented Lagrangian method for minimizing a (nonlinear) objective subject to general (nonlinear) constraints and simple bounds.

`QPB` is a primal-dual interior-point trust-region method for minimizing a general quadratic objective function over a polyhedral region.

`QPA` is a active/working-set method for minimizing a general quadratic objective function over a polyhedral region.

`LSQP` is an interior-point method for minimizing a linear or separable convex quadratic function over a polyhedral region.

`PRESOLVE` is a method for preprocessing linear and quadratic programming problems prior to solution by other packages.

`GLTR` is a method for minimizing a general quadratic objective function over the interior or boundary of a (scaled) hyper-sphere.

SILS provides an interface to the HSL sparse-matrix package MA27 that is functionally equivalent to the more recent HSL package HSL_MA57.

SCU uses a Schur complement update to find the solution of a sequence of linear systems for which the solution involving a leading sub-matrix may be found by other means.

In addition, GALAHAD contains the following auxiliary packages:

QPP reorders linear and quadratic programming problems to a convenient form prior to solution by other packages.

QPT provides a derived type for holding linear and quadratic programming problems.

SMT provides a derived type for holding sparse matrices in a variety of formats.

SORT gives implementations of both Quick- and Heap-sort methods

RAND provides pseudo-random numbers.

SPECFILE allows users to provide options to other packages, using lists of keyword-value pairs given in package-dependent option-specification files.

SYMBOLS assigns values to a list of commonly-used GALAHAD variables.

2.2 Details of major GALAHAD packages

2.2.1 LANCELOT B

The enlivened LANCELOT B offers a number of improvements over its predecessor, but alas is still far from the state-of-the-art. LANCELOT A is fully described in Conn et al. (1992), and the results of comprehensive tests are given in Conn, Gould and Toint (1996*a*). New features include

- the automatic allocation of workspace,
- a non-monotone descent strategy (see Toint, 1997, and Conn, Gould and Toint, 2000*a*, §10.1) to be used by default,
- the optional use of Moré and Toraldo (1991)-type projections (see also Lin and Moré, 1999*a*) during the subproblem solution phase,
- an interface to Lin and Moré's (1999*b*) public domain incomplete Cholesky factorization package ICFS for use as a preconditioner, and
- the optional use of structured trust regions to better model structured problems (see Conn, Gould, Sartenaer and Toint, 1996*b*, and Conn et al., 2000*a*, §10.2), and
- more flexibility over the choice of derivatives—derivatives need only be provided for a subset of the element functions from which the problem is built, the remainder being estimated by differences or secant approximations.

The main reason for extending LANCELOT's life is as a prototype for what may be achieved using Fortran 90/95 in preparation for future GALAHAD SQP solvers, since the problem data structure and interface is unlikely to change significantly.

To illustrate the effects of the new features, both LANCELOT B (using a number of these options) and LANCELOT A were run on all the examples (except linear and quadratic programs) from the CUTEr (see Gould, Orban and Toint, 2002*a*) test set—the CUTEr set differs from its predecessor in CUTE (see Bongartz, Conn, Gould and Toint, 1995) both in the number of problems, and by virtue of a long-overdue increase in default dimensions for all variable-dimensional problems. The options new to LANCELOT B that we considered were as follows:

- The default (in which a non-monotone descent strategy, with a history length of 1, a band preconditioner, with semi-bandwidth 5, exact second derivatives) is used.
- The default, except that a monotone descent strategy is used.
- The default, except that SR1 approximations to the second derivatives are used.
- The default, except that the Lin and Moré's (1999*b*) incomplete Cholesky factorization preconditioner, ICFS, with 5 extra work vectors, is used.
- the default, except that the Moré and Toraldo (1991) projected search, with 5 restarts, is used.
- The default, except that a structured trust region (see Conn et al., 1996*b*), is used.
- The default, except that the history length for the non-monotone descent strategy is increased to 5, is used.

LANCELOT A was run with its defaults, since an exhaustive test of other LANCELOT A options has already been performed (see Conn et al., 1996*a*).

In Figure 2.1 the performance profiles (see Dolan and Moré, 2001) for the CPU time (in seconds) and the numbers of function evaluations from these tests are reported. All of our experiments were performed on a single processor of a Compaq AlphaServer DS20 with 3.5 Gbytes of RAM, using the Compaq f90 compiler with full, machine specific, optimization, and each run was terminated unsuccessfully after 30 minutes or after 10000 function evaluations. Of the 749 problems thus considered (and for the best of the options), 151 (roughly 20%) were terminated due to excessive CPU times/function calls or because the evaluation of problem functions lead to floating-point exceptions—of those requiring more than 30 minutes/10000 calls, most could ultimately be solved by increasing the CPU and iteration limits.

Suppose that a given algorithm i from a competing set $\mathcal{A}$ reports a statistic $s_{ij} \geq 0$ when run on example j from our test set $\mathcal{T}$, and that the smaller this statistic the better the algorithm is considered. Let

$$k(s, s^*, \sigma) = \begin{cases} 1 & \text{if } s \leq \sigma s^* \\ 0 & \text{otherwise.} \end{cases}$$

Then, the *performance profile* of algorithm i is the function

$$p_i(\sigma) = \frac{\sum_{j \in \mathcal{T}} k(s_{i,j}, s_j^*, \sigma)}{|\mathcal{T}|} \quad \sigma \geq 1,$$

where $s_j^* = \min_{i \in \mathcal{A}} s_{ij}$. Thus $p_i(1)$ gives the fraction of the number of examples for which algorithm i was the most effective (according to statistics s_{ij}), $p_i(2)$ gives the fraction of the number for which algorithm i is within a factor of 2 of the best, and $\lim_{\sigma \longrightarrow \infty} p_i(\sigma)$ gives the fraction of the examples for which the algorithm

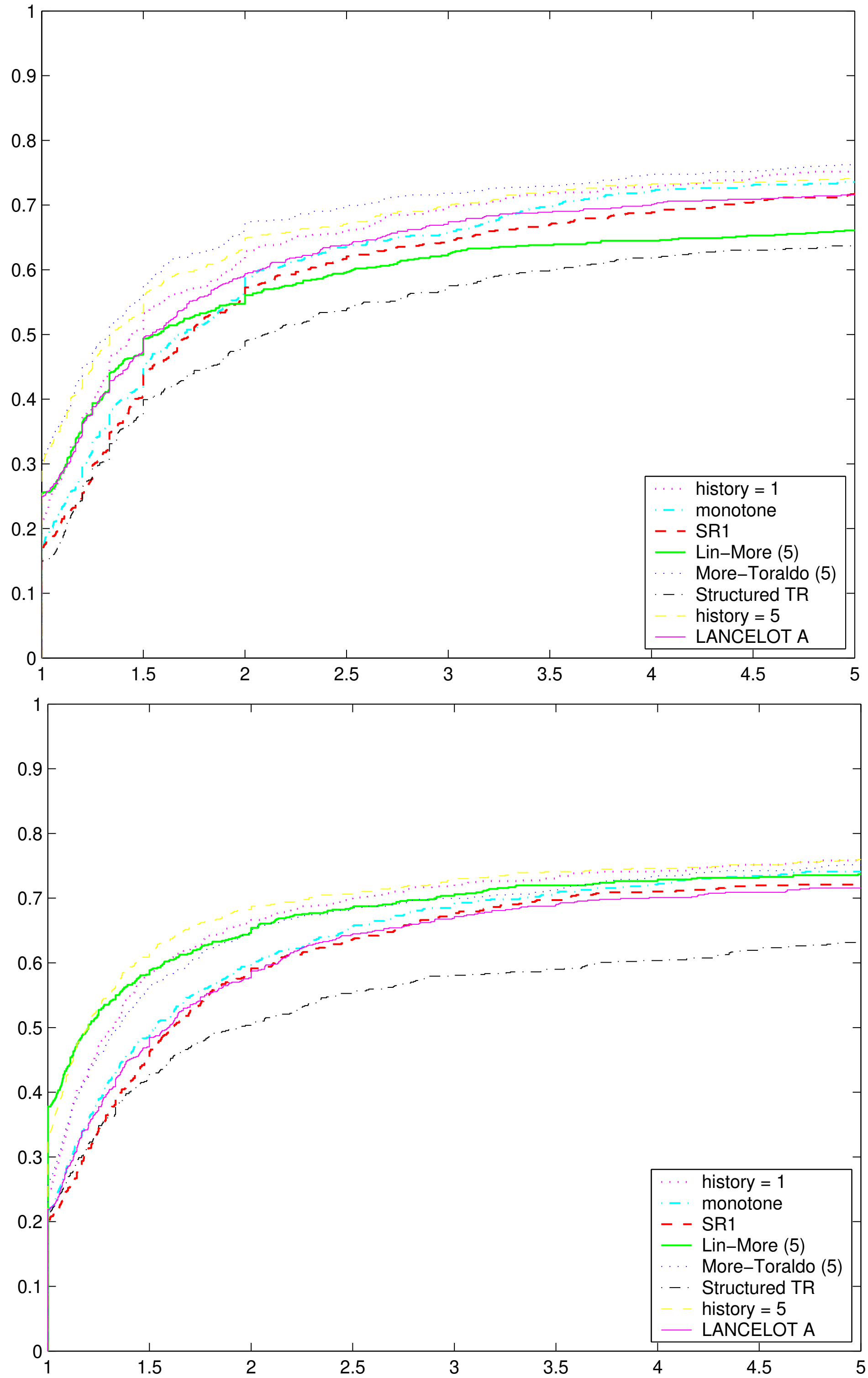

Figure 2.1: Performance profile for LANCELOT options: CPU times (top) and number of function evaluations (bottom). The horizontal axis is gives the argument σ, while the vertical axis records $p_i(\sigma)$ for each of the competing options, i.

succeeded. We consider such a profile to be a very effective means of comparing the relative merits of the new options available in LANCELOT B.

The benefits of the non-monotone and Lin-Moré strategies are apparent both in terms of CPU time and function evaluation reductions. In addition, we are pleased to see that the best of the new options show some gain with respect to LANCELOT A, particularly as we were initially concerned that moving from Fortran 77 to 90 might give rise to some performance penalties. The only new option that we are disappointed with is the use of a structured trust region, and, on the basis of these tests, we cannot really recommend this strategy. The full set of results are available as a internal report (see Gould, Orban and Toint, 2002*b*).

2.2.2 `QPB`

The module `QPB` is an implementation of a feasible-interior-point primal-dual trust-region method for quadratic programming.

To set the scene, the *quadratic programming* problem is to

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad q(x) \equiv \tfrac{1}{2}x^T Hx + g^T x \tag{2.1}$$

subject to the general linear constraints

$$c_i^l \leq a_i^T x \leq c_i^u, \quad i = 1, \ldots, m, \tag{2.2}$$

and the simple bound constraints

$$x_j^l \leq x_j \leq x_j^u, \quad j = 1, \ldots, n. \tag{2.3}$$

for given vectors g, a_i, c^l, c^u, x^l, x^u and a given symmetric (but not necessarily definite) matrix H—equality constraints and fixed variables are allowed by setting $c_i^u = c_i^l$ and $x_j^u = x_j^l$ as required, and any or all of the constraint bounds may be infinite. The required solution x to (2.1)–(2.3) necessarily satisfies the primal optimality conditions

$$Ax = c \tag{2.4}$$

and

$$c^l \leq c \leq c^u, \;\; x^l \leq x \leq x^u, \tag{2.5}$$

the dual optimality conditions

$$Hx + g = A^T y + z, \;\; y = y^l + y^u \;\text{ and }\; z = z^l + z^u, \tag{2.6}$$

and

$$y^l \geq 0, \;\; y^u \leq 0, \;\; z^l \geq 0 \;\text{ and }\; z^u \leq 0, \tag{2.7}$$

and the complementary slackness conditions

$$(Ax - c^l)^T y^l = 0, \;\; (Ax - c^u)^T y^u = 0, \;\; (x - x^l)^T z^l = 0 \;\text{ and }\; (x - x^u)^T z^u = 0, \tag{2.8}$$

where the vectors y and z are known as the Lagrange multipliers for the general linear constraints, and the dual variables for the bounds, respectively, and where the vector inequalities hold componentwise.

Primal-dual interior point methods iterate towards a point that satisfies the optimality conditions (2.4)–(2.8) by ultimately aiming to satisfy (2.4), (2.6) and (2.8), while ensuring that (2.5) and (2.7) are satisfied as strict inequalities at each stage. Appropriate norms of the amounts by which (2.4), (2.6) and (2.8) fail to be satisfied are known as the primal and dual infeasibility, and the violation of complementary slackness, respectively. The fact that (2.5) and (2.7) are satisfied as strict inequalities gives such methods their name, interior-point methods.

The problem is solved in two phases. The goal of the first "initial feasible point" phase is to find a strictly interior point which is primal feasible, that is that (2.4) is satisfied. The GALAHAD package `LSQP` (see §2.2.4) is used for this purpose, and offers the options of either accepting the first strictly feasible point found, or preferably of aiming for the so-called "analytic center" of the feasible region. Having found such a suitable initial feasible point, the second "optimality" phase ensures that (2.4) remains satisfied while iterating to satisfy dual feasibility (2.6) and complementary slackness (2.8). The optimality phase proceeds by approximately minimizing a sequence of barrier functions

$$q(x) - \mu \left[\sum_{i=1}^{m} \log(c_i - c_i^l) + \sum_{i=1}^{m} \log(c_i^u - c_i) + \sum_{j=1}^{n} \log(x_j - x_j^l) + \sum_{j=1}^{n} \log(x_j^u - x_j) \right],$$

for an appropriate sequence of positive barrier parameters μ converging to zero while ensuring that (2.4) remains satisfied and that x and c are strictly interior points for (2.5). Note that terms in the above summations corresponding to infinite bounds are ignored, and that equality constraints are treated specially.

Each of the barrier subproblems is solved using a trust-region method. Such a method generates a trial correction step $(\delta x, \delta c)$ to the current iterate (x, c) by replacing the nonlinear barrier function locally by a suitable quadratic model, and approximately minimizing this model in the intersection of (2.4) and a trust region $\|(\delta x, \delta c)\| \leq \Delta$ for some appropriate strictly positive trust-region radius Δ and norm $\|\cdot\|$. The step is accepted/rejected and the radius adjusted on the basis of how accurately the model reproduces the value of barrier function at the trial step. If the step proves to be unacceptable, a linesearch is performed along the step to obtain an acceptable new iterate. In practice, the natural primal "Newton" model of the barrier function is almost always less successful than an alternative primal-dual model, and consequently the primal-dual model is usually to be preferred.

The trust-region subproblem is approximately solved using the combined conjugate-gradient/Lanczos method (see Gould, Hribar and Nocedal, 2001 and Gould, Lucidi, Roma and Toint, 1999*a*) implemented in the GALAHAD module `GLTR` (see §2.2.6). Such a method requires a suitable preconditioner, and in our case, the only flexibility we have is in approximating the model of the Hessian. Although using a fixed form of preconditioning is sometimes effective, we have provided the option of an automatic choice, that aims to balance the cost of applying the preconditioner against the needs for an accurate solution of the trust-region subproblem. The preconditioner is applied using the GALAHAD factorization package `SILS` (see §2.2.7)—or optionally using `HSL_MA57` from HSL if this is available—but options at this stage are to factorize the preconditioner as a whole (the so-called "augmented system" approach), or to perform a block elimination first (the "Schur-complement" approach). The latter is usually to be preferred when a (non-singular) diagonal preconditioner is used, but may be inefficient if any of the columns of A is too dense.

The theoretical justification of the overall scheme, for problems with general objectives and inequality constraints, is given by Conn, Gould, Orban and Toint (2000*b*), in which we also present numerical results that suggest that it is indeed able to solve some problems of the size we had been aiming for. More recently, we investigated the ultimate rate of convergence of such schemes, and have shown that, under fairly general conditions, a componentwise superlinear rate is achievable both for quadratic and general nonlinear programs (see Gould, Orban, Sartenaer and Toint, 1999*b*).

Full advantage is taken of any zero coefficients in the matrix H or the vectors a_i. An older version of `QPB` (using `HSL_MA57` rather than `SILS`, see §2.2.7, but without some of the most recent features in `QPB`) is available commercially as `HSL_VE12` within HSL.

2.2.3 `QPA`

The module `QPA` is an implementation of a second approach to quadratic programming, this time of the active/working set variety. Although general in scope, `QPA` is intended within GALAHAD to deal with the case where a good estimate of the optimal active set has been determined (and thus that relatively few iterations will be required). The method is actually more general in scope, and is actually geared toward solving ℓ_1*-quadratic programming* problems of the form

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad q(x) + \rho_g v_g(x) + \rho_b v_b(x) \tag{2.9}$$

involving the quadratic objective $q(x)$ and the infeasibilities

$$v_g(x) = \sum_{i=1}^{m} \max(c_i^l - a_i^T x, 0) + \sum_{i=1}^{m} \max(a_i^T x - c_i^u, 0)$$

and

$$v_b(x) = \sum_{j=1}^{n} \max(x_j^l - x_j, 0) + \sum_{j=1}^{n} \max(x_j - x_j^u, 0).$$

At the k-th iteration of the method, an improvement to the value of the merit function $m(x, \rho_g, \rho_b) = q(x) + \rho_g v_g(x) + \rho_b v_b(x)$ at $x = x^{(k)}$ is sought. This is achieved by first computing a search direction $s^{(k)}$, and then setting $x^{(k+1)} = x^{(k)} + \alpha^{(k)} s^{(k)}$, where the stepsize $\alpha^{(k)}$ is chosen as the first local minimizer of $\phi(\alpha) = m(x^{(k)} + \alpha s^{(k)}, \rho_g, \rho_b)$ as α increases from zero. The stepsize calculation is straightforward, and exploits the fact that $\phi(\alpha)$ is a piecewise quadratic function of α.

The search direction is defined by a subset of the "active" terms in $v(x)$, i.e., those for which $a_i^T x = c_i^l$ or c_i^u (for $i = 1, \ldots, m$) or $x_j = x_j^l$ or x_j^u (for j=1,…,n). The "working" set $W^{(k)}$ is chosen from the active terms, and is such that its members have linearly independent gradients. The search direction $s^{(k)}$ is chosen as an approximate solution of the equality-constrained quadratic program

$$\underset{s \in \mathbb{R}^n}{\text{minimize}} \quad q(x^{(k)} + s) + \rho_g l_g^{(k)}(s) + \rho_b l_b^{(k)}(s), \tag{2.10}$$

subject to

$$a_i^T s = 0, \;\; i \in \{1, \ldots, m\} \cap W^{(k)}, \;\; \text{and} \;\; x_j = 0, \;\; i \in \{1, \ldots, n\} \cap W^{(k)}, \tag{2.11}$$

where

$$l_g^{(k)}(s) = - \sum_{\substack{i=1 \\ a_i^T x < c_i^l}}^{m} a_i^T s \; + \sum_{\substack{i=1 \\ a_i^T x > c_i^u}}^{m} a_i^T s$$

and

$$l_b^{(k)}(s) = - \sum_{\substack{j=1 \\ x_j < x_j^l}}^{n} s_j \; + \sum_{\substack{j=1 \\ x_j > x_j^u}}^{n} s_j .$$

The equality-constrained quadratic program (2.10)–(2.11) is solved by a projected preconditioned conjugate gradient method (see Gould et al., 2001). The method terminates either after a pre-specified number of iterations, or if the solution is found, or if a direction of infinite descent, along which $q(x^{(k)}+s)+\rho_g l_g^{(k)}(s)+\rho_b l_b^{(k)}(s)$ decreases without bound within the feasible region (2.11), is located. Successively more accurate approximations are required as suspected solutions of (2.9) are approached.

Preconditioning of the conjugate gradient iteration requires the solution of one or more linear systems of the form

$$\begin{pmatrix} M^{(k)} & A^{(k)T} \\ A^{(k)} & 0 \end{pmatrix} \begin{pmatrix} p \\ u \end{pmatrix} = \begin{pmatrix} g \\ 0 \end{pmatrix}, \tag{2.12}$$

where $M^{(k)}$ is a "suitable" approximation to H and the rows of $A^{(k)}$ comprise the gradients of the terms in the current working set. Rather than recomputing a factorization of the preconditioner at every iteration, a Schur complement method is used, recognising the fact that gradual changes occur to successive working sets. The main iteration is divided into a sequence of "major" iterations. At the start of each major iteration (say, the overall iteration l), a factorization of the current "reference" matrix, that is the matrix

$$\begin{pmatrix} M^{(l)} & A^{(l)T} \\ A^{(l)} & 0 \end{pmatrix} \tag{2.13}$$

is obtained using the GALAHAD factorization package `SILS` (see §2.2.7)—or, once again, optionally using `HSL_MA57` from HSL if this is available. This reference matrix may be factorized as a whole, using the augmented system approach, or by performing a block elimination first, using the "Schur-complement" approach. The latter is usually to be preferred when a (non-singular) diagonal preconditioner is used, but may be inefficient if any of the columns of $A^{(l)}$ is too dense. Subsequent iterations within the current major iteration obtain solutions to (2.12) via the factors of (2.13) and an appropriate (dense) Schur complement, obtained from `SCU` in GALAHAD (see §2.2.8). The major iteration terminates once the space required to hold the factors of the (growing) Schur complement exceeds a given threshold.

The working set changes by (a) adding an active term encountered during the determination of the step-size, or (b) the removal of a term if $s = 0$ solves (2.10)–(2.11). The decision on which to remove in the latter case is based upon the expected decrease upon the removal of an individual term, and this information is available from the magnitude and sign of the components of the auxiliary vector u computed in (2.12). At optimality, the components of u for a_i terms will all lie between 0 and ρ_g—and those for the other terms between 0 and ρ_b—and any violation of this rule indicates further progress is possible.

To solve quadratic programs of the form (2.1)–(2.3), a sequence of problems of the form (2.9) are solved, each with a larger value of ρ_g and/or ρ_b than its predecessor. The required solution has been found once the infeasibilities $v_g(x)$ and $v_b(x)$ have been reduced to zero at the solution of (2.9) for the given ρ_g and ρ_b.

Having proposed and implemented two very different quadratic programming methods, an obvious question is: how do the methods compare. We examined this question in Gould and Toint (2001) by comparing `QPA` and `QPB` on the CUTEr (see Gould et al., 2002*a*) QP test set.

While for modest sized problems, started from "random" points, the two methods are roughly comparable, the advantages of the interior-point approach become quite clear when problem dimensions increase. For problems involving tens of thousands of unknowns and/or constraints, our active set approach simply takes too many iterations, while the number of iterations required by the interior point approach seems relatively insensitive to dimension size. For general problems involving hundreds of thousands or even millions of unknowns/constraints, the active set approach is impractical, while we illustrate in Table 2.1 that `QPB` is able to solve problems of this size. `QPB` also appears to scale well with dimension, as can be seen see in Table 2.2.

While such figures might seem to indicate that `QPB` should always be preferred to `QPA`, this is not the case. In particular, if a good estimate of the solution—more particularly, the optimal active set— is known, active-set methods may exploit this, while interior-point methods are (currently) less able to do so. In particular Gould and Toint (2001) illustrate that `QPA` often outperforms `QPB` on warm-started problems unless the problem is (close to) degenerate or very ill conditioned. Thus, since nonlinear optimization (SQP) algorithms often solve a sequence of related problems for which the optimal active sets are almost or actually identical, there is good reason to maintain both `QPA` and `QPB` in GALAHAD.

Name	n	m	type	its	time
QPBAND	100000	50000	C	13	157
QPBAND	200000	100000	C	17	1138
QPBAND	400000	200000	C	17	2304
QPBAND	500000	250000	C	17	2909
QPNBAND	100000	50000	NC	12	32
QPNBAND	200000	100000	NC	13	71
QPNBAND	400000	200000	NC	14	156
QPNBAND	500000	250000	NC	13	181

Table 2.1: QPB on large QP examples. Runs performed on a Compaq AlphaServer DS20 (3.5 Gbytes RAM), time in CPU seconds. n is the number of unknowns, and m is the number of general constraints. C indicates a convex problem, while NC is a convex one.

Name	n	m	type	its	time
PORTSQP	10	1	C	11	0.02
PORTSQP	100	1	C	15	0.03
PORTSQP	1000	1	C	26	0.09
PORTSQP	10000	1	C	37	1.26
PORTSQP	100000	1	C	20	9.48
PORTSQP	1000000	1	C	11	72.31
PORTSNQP	10	2	NC	21	0.03
PORTSNQP	100	2	NC	30	0.04
PORTSNQP	1000	2	NC	39	0.17
PORTSNQP	10000	2	NC	32	1.70
PORTSNQP	100000	2	NC	107	58.69
PORTSNQP	1000000	2	NC	22	209.53

Table 2.2: How QPB scales with dimension. Notation as for Table 2.1.

An enhanced version of QPA (using HSL_MA57 rather than SILS, see §2.2.7) is available commercially as HSL_VE19 within HSL.

2.2.4 LSQP

LSQP is an interior-point method for minimizing a linear or separable convex quadratic function

$$\text{minimize} \;\; \tfrac{1}{2}\sum_{i=1}^{n} w_i^2 (x_i - x_i^0)^2 + g^T x,$$

for given weights w and gradient g, over the polyhedral region (2.2)–(2.3). In the special case where $w = 0$ and $g = 0$ the so-called analytic center of the feasible set will be found, while linear programming, or constrained least distance, problems may be solved by picking $w = 0$, or $g = 0$, respectively. The basic algorithm is that of Zhang (1994), and is quite standard. LSQP is used within GALAHAD by QPB.

Note that since predictor-corrector steps are not taken, the method is unlikely to be as efficient as state-of-the-art interior-point methods for linear programming. We intend to remedy this defect for Version 2.

2.2.5 `PRESOLVE`

The module `PRESOLVE` is intended to pre-process quadratic programming problems. The purpose of pre-processing (or presolving) quadratic programming problems is to exploit the optimality equations (2.4)–(2.8) so as to both simplify the problem and reduce the problem to a standard form (that makes subsequent manipulation easier), defined as follows:

- The variables are ordered so that their bounds appear in the order

$$\begin{array}{lcccccc} \text{free} & & & x_j & & \\ \text{non-negativity} & 0 & \leq & x_j & & \\ \text{lower} & x_j^l & \leq & x_j & & \\ \text{range} & x_j^l & \leq & x_j & \leq & x_j^u \\ \text{upper} & & & x_j & \leq & x_j^u \\ \text{non-positivity} & & & x_j & \leq & 0 \end{array}$$

  Fixed variables are removed. Within each category, the variables are further ordered so that those with non-zero diagonal Hessian entries occur before the remainder.

- The constraints are ordered so that their bounds appear in the order

$$\begin{array}{lccccc} \text{non-negativity} & 0 & \leq & (Ax)_i & & \\ \text{equality} & c_i^l & = & (Ax)_i & & \\ \text{lower} & c_i^l & \leq & (Ax)_i & & \\ \text{range} & c_i^l & \leq & (Ax)_i & \leq & c_i^u \\ \text{upper} & & & (Ax)_i & \leq & c_i^u \\ \text{non-positivity} & & & (Ax)_i & \leq & 0 \end{array}$$

  Free constraints are removed.

- In addition, constraints may be removed or bounds tightened, to reduce the size of the feasible region or simplify the problem if this is possible, and bounds may be tightened on the dual variables and the multipliers associated with the problem.

The presolving algorithm (see Gould and Toint, 2002) proceeds by applying a (potentially long) series of simple transformations to the problem, each transformation introducing a further simplification of the problem. These involve the removal of empty and singleton rows, the removal of redundant and forcing primal constraints, the tightening of primal and dual bounds, the exploitation of linear singleton, linear doubleton and linearly unconstrained columns, merging of dependent variables, row "sparsification" and splitting equalities. Transformations are applied in successive passes, each pass involving the following actions:

1. remove empty and singletons rows,
2. try to eliminate variables that are linearly unconstrained,
3. attempt to exploit the presence of linear singleton columns,
4. attempt to exploit the presence of linear doubleton columns,
5. complete the analysis of the dual constraints,
6. remove empty and singletons rows,

7. possibly remove dependent variables,
8. analyze the primal constraints,
9. try to make A sparser by combining its rows,
10. check the current status of the variables, dual variables and multipliers.

All these transformations are applied on the structure of the original problem, which is only permuted to standard form after all transformations are completed. The reduced problem may then solved by a quadratic or linear programming solver. Finally, the solution of the simplified problem is re-translated to the variables/constraints format of the original problem in a "restoration" phase.

At the overall level, the presolving process follows one of the following two sequences:

$$\boxed{\text{initialize}} \rightarrow \left[\boxed{\text{apply transformations}} \rightarrow \text{(solve problem)} \rightarrow \boxed{\text{restore}} \right] \rightarrow \boxed{\text{terminate}}$$

or

$$\boxed{\text{initialize}} \rightarrow \left[\boxed{\begin{array}{l}\text{read}\\ \text{specfile}\end{array}} \rightarrow \boxed{\begin{array}{l}\text{apply}\\ \text{transformations}\end{array}} \rightarrow \begin{pmatrix}\text{solve}\\ \text{problem}\end{pmatrix} \rightarrow \boxed{\text{restore}} \right] \rightarrow \boxed{\text{terminate}}$$

where the procedure's control parameter may be modified by reading an external "specfile", and where (solve problem) indicates that the reduced problem is solved. Each of the "boxed" steps in these sequences corresponds to calling a specific routine of the package, while bracketed subsequence of steps means that they can be repeated with problems having the same structure.

An implementation is available as `PRESOLVE` in the GALAHAD library. Gould and Toint (2002) indicate that, when considering all 178 linear and quadratic programming problems in the CUTE (see Bongartz et al., 1995) test set, an average reduction of roughly 20% in both the number of unknowns and the number of constraints results from applying `PRESOLVE`. When applying the GALAHAD QP solver `QPB` (see §2.2.2), an overall average reduction of roughly 10% in CPU time results. In some cases, the gain is dramatic. For example, for the problems `GMNCASE4`, `STNPQ1`, `STNQP2` and `SOSQP1`, `PRESOLVE` removes all the variables and constraints, and thus reveals the complete solution to the problem without resorting to a QP solver.

Currently `PRESOLVE` is not embedded within `QPA/B` or `LSQP` and must be called separately, but we intend to correct this defect for Version 2.0.

2.2.6 `GLTR`

`GLTR` aims to find the global solution to the problem of minimizing the quadratic function (2.1) where the variables are required to satisfy the constraint $\|x\|_M \leq \Delta$, where the M-norm of x is $\|x\|_M = \sqrt{x^T M x}$ for some symmetric, positive definite M, and where $\Delta > 0$ is a given scalar. This problem commonly occurs as a trust-region subproblem in nonlinear optimization calculations, and is used within GALAHAD by `QPB`. The method may be suitable for large n as no factorization of H is required. Reverse communication is used to obtain matrix-vector products of the form Hz and $M^{-1}z$. The package may also be used to solve the related problem in which x is instead required to satisfy the **equality constraint** $\|x\|_M = \Delta$. The method is described in detail in Gould et al. (1999*a*), and `GLTR` is a slightly improved version of the HSL package `HSL_VF05`.

2.2.7 SILS

The module SILS provides a Fortran 90 interface to the Fortran 77 HSL sparse linear equation package MA27 (Duff and Reid 1982). The interface and functionality are designed to be identical to the more recent HSL Fortran 90 package HSL_MA57 (Duff 2001), enabling anyone with HSL_MA57 to easily substitute this for SILS throughout GALAHAD. The reason that we are forced to rely on MA27 rather than the superior HSL_MA57 by default is simply that the former is available without charge from the HSL Archive (http://hsl.rl.ac.uk/hslarchive,) while the latter is only available commercially. SILS (and hence wither MA27 or HSL_MA57) is required by QPA/B and LSQP, and is used optionally by LANCELOT B.

2.2.8 SCU

SCU may be used to find the solution to an extended system of $n + m$ sparse real linear equations in $n + m$ unknowns,

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

in the case where the n by n matrix A is nonsingular and solutions to the systems

$$Ax = b \text{ and } A^T y = c$$

may be obtained from an external source, such as an existing factorization. The subroutine uses reverse communication to obtain the solution to such smaller systems. The method makes use of the Schur complement matrix

$$S = D - CA^{-1}B.$$

The Schur complement is stored and factorized as a dense matrix and the subroutine is thus appropriate only if there is sufficient storage for this matrix. Special advantage is taken of symmetry and definiteness in the coefficient matrices. Provision is made for introducing additional rows and columns to, and removing existing rows and columns from, the extended matrix.

SCU is used by both LANCELOT B and QPA to cope with core linear systems that arise as constraints and variables enter and leave their active/working sets. A slightly simplified version of SCU is available in HSL as HSL_MA69.

3 Installation

Just as for its immediate predecessors CUTEr and SifDec (see Gould et al., 2002*a*), GALAHAD is designed to be used in a multi-platform, multi-compiler environment in which core (source and script) files are available in a single location, and machine/compiler/operating system specific components (most especially compiled libraries of binaries) are isolated in uniquely identifiable subdirectories. As before, we have concentrated on UNIX platforms, principally since we have no experience with other systems.

GALAHAD is provided as a series of files and directories, all lying beneath a root directory, which we shall refer to as $GALAHAD. The directory structure is illustrated in Figure 3.2.

Before installation the sub-directories objects, modules, makefiles, versions and bin/sys will all be empty. The installation script, install_galahad, prompts the user for the answers to a series of questions aimed at determining what machine type, operating system and compiler (from a large list of predefined possibilities encoded in the arch sub-directory) to build GALAHAD for—we call this combination of a machine, operating system and compiler an *architecture*. Each architecture is assigned a simple

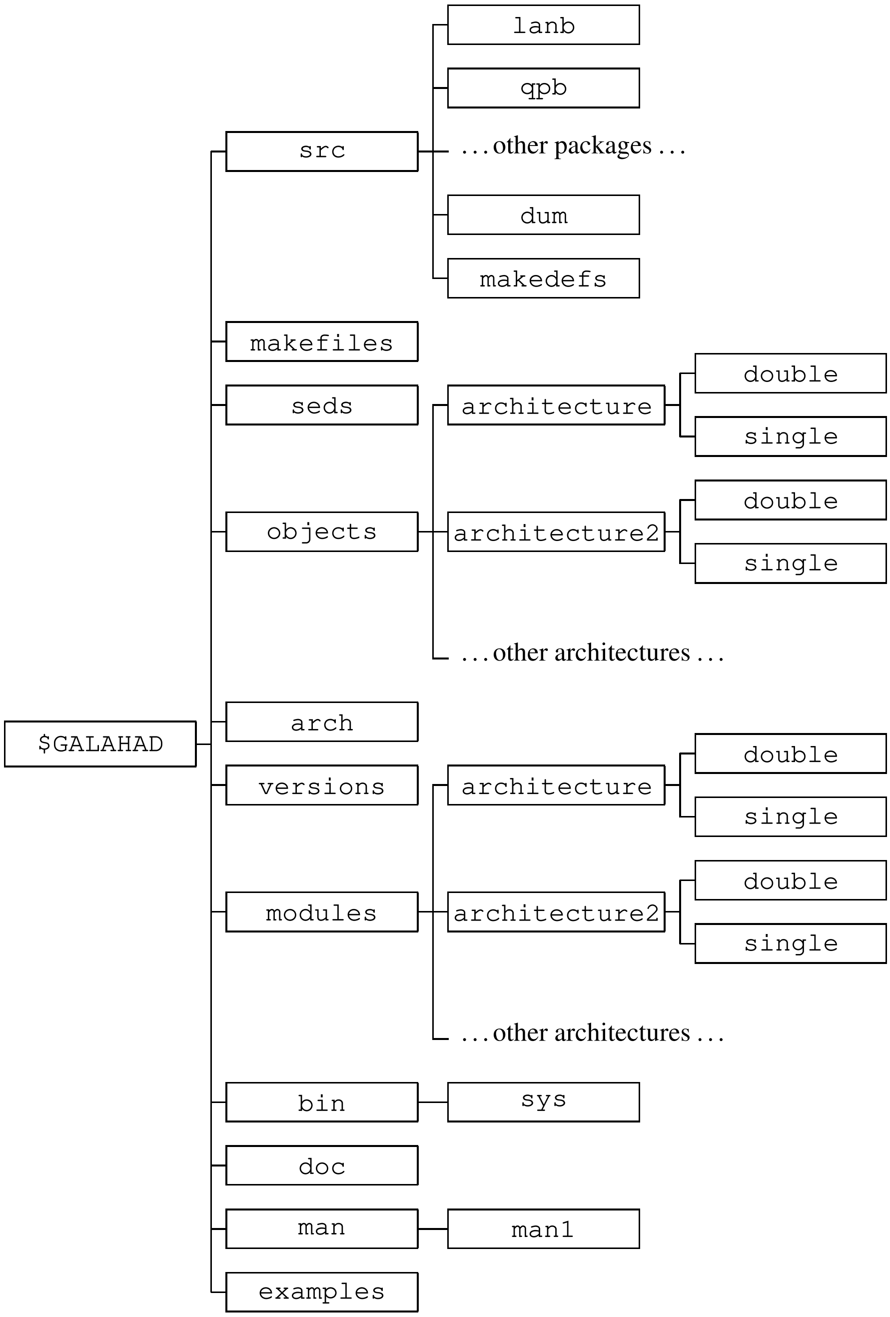

Figure 3.2: Structure of the GALAHAD directories

(mnemonic) architecture *code* name, say `architecture`—for example a version for the NAG Fortran 95 compiler on a PC runing Linux is coded `pc.lnx.n95`, while another for the Compaq Fortran 90 compiler on an Alpha system running Tru64 Unix is `alp.t64.f90`. Having determined the architecture, the installation script builds sub-directories of `objects` and `modules` named `architecture`, as well as further sub-directories `double` and `single` of these to hold architecture-dependent compiled libraries and module file information. In addition, architecture-dependent makefile information and environment variables for execution scripts are placed in files named `architecture` in the `makefiles` and `bin/sys` sub-directories, and a file recording how the code is related to the architecture is put in `versions`.

The Fortran source codes for each GALAHAD package occurs in a separate sub-directory of the `src` directory—the sub-directory contains the package source, a comprehensive test program (along with a simpler second test program, which is used as an illustration on how to call the package in the accompanying documentation), and a makefile. Since the order of compilation of Fortran modules is important, and as we have seen there is a strong interdependency between the GALAHAD packages, the makefiles have to be carefully crafted. For this reason (and unlike for CUTEr), we have chosen not to use tools such as `imake` to build and maintain the makefiles. Postscript and PDF Documentation for all packages is contained in `doc`.

Not every component of GALAHAD is distributed as part of the package. In particular, the HSL Archive code `MA27` must be downloaded prior to the installation from

```
http://hsl.rl.ac.uk/archive/hslarchive.html
```

before any of the QP packages can be used (it is optionally used by LANCELOT B). Additionally GALAHAD makes optional use of the Lin-Moré preconditioner `ICFS`, available as part of the MINPACK 2 library, via

```
http://www-unix.mcs.anl.gov/~more/icfs/
```

and the HSL codes `MA57` and `MC61` which are only available commercially. If any non-default code is used, the file `packages` in the directory `src/makedefs` must be edited to describe where the external code may be found—details are given in `packages`. Default dummy versions of all optional codes are provided in `src/dum` to ensure that linking prior to execution works properly.

Once the correct directory structure is in place, the installation script builds a random library of the required precision by visiting each of the sub-directories of `src` and calling the Unix utility `make`. GALAHAD packages are all written in double precision, but if a user prefers to use single precision, the makefiles call suitable Unix `sed` scripts (stored in `seds`) to transform the source prior to compilation. A user may choose to install all of GALAHAD, or just the QP or LANCELOT B components. There are also `sed` features to automatically switch from `SILS` (see §2.2.7) to `MA57` if the latter is available. The make option `make tests` runs comprehensive tests of all compiled components.

4 Interfaces to the CUTEr test set

As well as being stand-alone Fortran modules, we provide transparent interfaces between LANCELOT B, `QPA/B`, `LSQP` and `PRESOLVE` and problems written in the Standard Input Format (SIF, see Conn et al., 1992), most particularly the CUTEr (see Gould et al., 2002*a*) test set. To use these interfaces LANCELOT B users will need to have installed the package SifDec, while users wishing to use the interfaces to the QP packages will additionally need CUTEr (see Gould et al., 2002*a*).

To run one of the supported packages on an example stored in, say, `EXAMPLE.SIF`, a user needs to issue the command

```
sdgal code package EXAMPLE.[SIF]
```

where `code` is the architecture code discussed in §3, `package` defines the package to be used—it may be one of `lanb`, `qpa`, `qpb` or `pre`, with access to `LSQP` provided via `qpb`—and the suffix `[.SIF]` is optional. This command translates the SIF file into Fortran subroutines and related data using the decoder provided in SifDec, and then calls the required optimization package to solve the problem. Once a problem has been decoded, it may be re-called (perhaps with different options) using the auxiliary command

```
gal code package
```

A few SIF examples are given in the `examples` sub-directory, while the `sdgal` and `gal` commands are in the `bin` sub-directory, and have man-page descriptions in the `man/man1` sub-directory.

One additional feature that has been added is that if the user has access to the HSL automatic differentiation packages `HSL_AD01` or `HSL_AD02` (see Pryce and Reid, 1998), these may be used to generate automatic first and second derivatives for the element and group function (see Conn et al., 1992) from which the overall problem is re-assembled by CUTEr and LANCELOT B. Of course, ideally, it would be better to use one of the more commonly-occurring packages such as ADIFOR (see Bischof, Carle, Corliss, Griewank and Hovland, 1992), and we plan to do this in the future.

As with LANCELOT A, options may be passed directly to the solvers by means of user-definable option-specification files. Each SIF interface has its own set of options, but overall control is maintained via the GALAHAD module `SPECFILE`.

5 Availability

GALAHAD may be downloaded from its WWW site,

```
http://galahad.rl.ac.uk/galahad-www
```

There are restrictions on commercial use, and all users are required to agree to a number of minor conditions of use.

6 Conclusions

We have described the scope and design of the first release of GALAHAD, a library of Fortran 90 packages for nonlinear optimization. Version 1.0 of the library particularly addresses quadratic programming problems, although there is an updated version of LANCELOT for more general problems. The package may be downloaded from its WWW site, `http://galahad.rl.ac.uk/galahad-www`.

In future, we intend to use the quadratic programming packages as the basic tool within one or more SQP methods for nonlinear optimization. We are currently developing AMPL (see Fourer, Gay and Kernighan, 1993) interfaces for the principal packages so that users will be able to use a more natural interface than provided by SIF. In addition, we plan to incorporate the preprocessing tools directly into the QP solvers, rather than having them stand-alone as is currently the case.

References

H. Benson, D. F. Shanno, and R. J. Vanderbei. A comparative study of large-scale nonlinear optimization algorithms. Technical Report ORFE 01-04, Operations Research and Financial Engineering, Princeton University, New Jersey, USA, 2001.

Chr. Bischof, A. Carle, G. Corliss, A. Griewank, and P. Hovland. ADIFOR - generating derivative codes from fortran programs. *Scientific Programming*, **1**, 1–29, 1992.

I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: Constrained and Unconstrained Testing Environment. *ACM Transactions on Mathematical Software*, **21**(1), 123–160, 1995.

R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large scale nonlinear programming. *SIAM Journal on Optimization*, **9**(4), 877–900, 1999.

C. M. Chin. Numerical results of SLPSQP, filterSQP and LANCELOT on selected CUTE test problems. Numerical Analysis Report NA/203, Department of Mathematics, University of Dundee, Scotland, 2001.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. LANCELOT*: a Fortran package for Large-scale Nonlinear Optimization (Release A)*. Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, New York, 1992.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization. *Mathematical Programming, Series A*, **73**(1), 73–110, 1996*a*.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-region methods*. SIAM, Philadelphia, 2000*a*.

A. R. Conn, N. I. M. Gould, D. Orban, and Ph. L. Toint. A primal-dual trust-region algorithm for non-convex nonlinear programming. *Mathematical Programming*, **87**(2), 215–249, 2000*b*.

A. R. Conn, N. I. M. Gould, A. Sartenaer, and Ph. L. Toint. Convergence properties of minimization algorithms for convex constraints using a structured trust region. *SIAM Journal on Optimization*, **6**(4), 1059–1086, 1996*b*.

J. Czyzyk, M. P. Mesnier, and J. J. Moré. The NEOS Server,. *IEEE Journal on Computational Science and Engineering*, **5**, 68–75, 1998.

E. D. Dolan and J. J. Moré. Benchmarking optimization software with cops. Technical Report ANL/MCS-246, Argonne National Laboratory, Illinois, USA, 2000.

E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. Technical Report ANL/MCS-P861-1200, Argonne National Laboratory, Illinois, USA, 2001.

I. S. Duff. MA57 - a new code for the solution of sparse symmetric indefinite systems. Technical Report (in preparation), Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2001.

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.

R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, **91**(2), 239–269, 2002.

R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A modelling language for mathematical programming*. The Scientific Press, San Fransisco, California, USA, 1993.

P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, **(to appear)**, 2002.

N. I. M. Gould and Ph. L. Toint. Numerical methods for large-scale non-convex quadratic programming. Technical Report RAL-TR-2001-017, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2001.

N. I. M. Gould and Ph. L. Toint. Preprocessing for quadratic programming. Technical Report RAL-TR-2002-001, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2002.

N. I. M. Gould, M. E. Hribar, and J. Nocedal. On the solution of equality constrained quadratic problems arising in optimization. *SIAM Journal on Scientific Computing*, **23**(4), 1375–1394, 2001.

N. I. M. Gould, S. Lucidi, M. Roma, and Ph. L. Toint. Solving the trust-region subproblem using the Lanczos method. *SIAM Journal on Optimization*, **9**(2), 504–525, 1999*a*.

N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr (and SifDec), a constrained and unconstrained testing environment, revisited. Technical Report RAL-TR-2002-009, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2002*a*.

N. I. M. Gould, D. Orban, and Ph. L. Toint. Results from a numerical evaluation of LANCELOT B . Numerical Analysis Group Internal Report 2002-1, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2002*b*.

N. I. M. Gould, D. Orban, A. Sartenaer, and Ph. L. Toint. Superlinear convergence of primal-dual interior-point algorithms for nonlinear programming. *SIAM Journal on Optimization*, **11**(4), 974–1002, 1999*b*.

W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes*. Number 187 *In* 'Lecture Notes in Economics and Mathematical Systems'. Springer Verlag, Heidelberg, Berlin, New York, 1981.

HSL. A collection of Fortran codes for large-scale scientific computation, 2002. See http://www.cse.clrc.ac.uk/Activity/HSL.

C. Lin and J. J. Moré. Incomplete Cholesky factorizations with limited memory. *SIAM Journal on Scientific Computing*, **21**(1), 24–45, 1999*a*.

C. Lin and J. J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, **9**(4), 1100–1127, 1999*b*.

J. J. Moré and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIAM Journal on Optimization*, **1**(1), 93–113, 1991.

J. D. Pryce and J. K. Reid. `AD01`, a Fortran 90 code for automatic differentiation. Technical Report RAL-TR-1998-057, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 1998.

Ph. L. Toint. A non-monotone trust-region algorithm for nonlinear optimization subject to convex constraints. *Mathematical Programming*, **77**(1), 69–94, 1997.

R. J. Vanderbei and D. F. Shanno. An interior point algorithm for nonconvex nonlinear programming. *Computational Optimization and Applications*, **13**, 231–252, 1999.

Y. Zhang. On the convergence of infeasible interior-point methods for the horizontal linear complementarity problem. *SIAM Journal on Optimization*, **4**(1), 208–227, 1994.