Some Linear Algebra Issues in Large-Scale Optimization

Margaret H. Wright
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

Abstract

Solving linear systems is a fundamental ingredient in numerical optimization. Particularly in large-scale optimization, these systems encompass a surprisingly wide variation not only in form and representation of matrices, but also in the level of solution accuracy. A broad and often vaguely specified concept in the latter context is "inexact" solution. For unconstrained, linearly constrained and nonlinearly constrained optimization in turn, we discuss selected features of large-scale problems and of the associated forms of inexactness. Various unresolved issues in each problem category are mentioned, along with recent areas of research activity stressing linear algebra.

This paper is based on an invited talk given at the NATO Advanced Study Institute "Linear Algebra for Large-Scale and Real-Time Applications", August 3–14, 1992, Leuven, Belgium, and will appear in the proceedings, published by Kluwer, 1993.

1. Introduction

An optimization problem involves finding the "best": given a specified measure of goodness (the *objective function*), we try to determine its optimal value by adjusting a set of variables that may be subject to restrictions or limitations (*constraints*). Linear algebra and optimization are closely related in many ways, of which we mention two:

- Optimization methods include linear algebraic subproblems (mostly, linear systems). Hence, the needs of optimization methods indicate areas of linear algebra in which improvements would be helpful.
- Efficient linear algebra techniques inspire optimization methods that use them. If researchers in optimization know that a certain kind of linear system can be solved quickly and stably, it is natural to develop optimization methods that include such systems. For example, the simplex method for linear programming and quasi-Newton methods for nonlinear optimization rely on the ability to perform low-rank updates to matrix factorizations.

A *large-scale* optimization problem cannot be given a precise general definition, since the notion of "size" depends on context. Broadly speaking, however, a "large" problem is one in which the *number of variables* (or constraints) noticeably affects our ability to solve the problem in the given computing environment. Confusion in discussing large-scale optimization arises because some authors restrict their definition of "large" problems to those in which the associated linear systems are *sparse*. Although sparsity does indeed characterize many, perhaps even a majority, of large-scale problems solved today, it is *not* a requirement. The recent surveys [6, 10] discuss many issues in large-scale optimization, and provide extensive bibliographies.

It is undeniable that larger and larger optimization problems are being solved successfully. In certain areas, such as linear programming, problems considered impossibly large only a few years ago are solved almost routinely in practical applications—sometimes in real time. The state of the art has advanced in part because of hardware: enormous gains in computer speed and storage, and increasing availability of high-performance computing environments in a variety of architectures. Progress in algorithms, however, has been even more significant.

For any particular large problem, issues such as efficient use of storage, organization of calculations, and adaptation to specialized architectures must be faced—for example, determining the best way to represent and solve partial differential equations that define an objective function. In addition, the same issues appear in a more general form as they apply to common linear algebraic procedures within optimization algorithms. The exigencies of linear algebra accordingly dictate attention to the inner workings of large-scale optimization methods.

Two themes will recur throughout this paper:

1. The special, interesting or important features of large-scale optimization problems and methods, particularly as they relate to linear algebra;

2. The role of *inexactness* in solving linear algebraic subproblems. (Although inexactness is frequently mentioned in the optimization literature, it is rarely carefully defined.)

2. Formulation

2.1. Problem statement

The objective function, which we assume is minimized, will be denoted by $f(x)$, where x is a real n-vector. Constraints occur in two varieties: equalities of the form $c_j(x) = 0$, and inequalities of the form $c_j(x) \geq 0$. (For convenience, we have arbitrarily chosen to express inequalities with a "greater than" relation.) The constraints may include a mixture of equalities and inequalities. The objective and constraint functions are assumed to be twice-continuously differentiable.

The gradient and Hessian matrix of $f(x)$ will be denoted by $g(x)$ and $H(x)$. The n-vector $a_j(x)$ is the gradient of $c_j(x)$, and $H_j(x)$ is its Hessian. The matrix $A(x)$ is defined as

$$A(x) = \begin{pmatrix} a_1^T(x) \\ \vdots \\ a_m^T(x) \end{pmatrix},$$

and is the usual Jacobian of the set of m constraints $\{c_j(x)\}$. (Note that the jth row of $A(x)$ is $a_j^T(x)$.) The generic solution of an optimization problem will be denoted by x^*, which is assumed to be only a local solution. For simplicity and brevity, we assume for each problem category that standard second-order sufficiency conditions hold at x^*. Details of these conditions are given in most textbooks on optimization—for example, [9, 10, 15, 24].

Constraint functions are often specialized beyond the general form above. A *linear constraint* has the form $a_j^T x = b_j$ or $a_j^T x \geq b_j$, where a_j is a constant vector and b_j is a scalar. The Jacobian matrix of a set of linear constraints is thus a constant matrix, and the Hessian of a linear constraint is zero. The simplest and probably the most frequent form of linear constraint imposes lower and/or upper bounds on a variable, i.e. $l_j \leq x_j \leq u_j$.

For large-scale problems, it can be advantageous to "remove" inequality constraints by transforming to so-called *standard form*, in which the only inequality constraints are bounds on the variables. To produce standard form, inequalities are replaced by equalities that include extra variables subject to bounds, so that the inequality $c_j(x) \geq 0$ is replaced by the equality $c_j(x) - y_j = 0$, where the new "slack" variable y_j must satisfy $y_j \geq 0$. See [15, 24] for a discussion of some of the benefits of standard form. The main advantage is that, for some algorithms, standard form ensures that certain matrices remain fixed in dimension throughout the optimization, thereby simplifying the linear algebra.

2.2. Relevant problem features

We have already mentioned that optimization problems are characterized by features such as the presence or absence of constraints, the linearity or nonlinearity of constraints, and the options of inequality or standard form for the constraints. Beyond these features, a major difference between small and large problems is the attention paid to variations connected with linear algebra.

For small optimization problems, the linear algebra costs for an iteration can typically be estimated from the problem dimensions, which bound the size of matrices involved. Consider unconstrained minimization using a Newton-based method for small n (see Section 3.1), which requires a Cholesky factorization of the dense $n \times n$ Hessian matrix at a cost of approximately $n^3/6$ flops. Even if a small problem has features that could be exploited to speed up the factorization (say, many elements of the Hessian are known to be zero or one), it would not generally be considered worthwhile to expend the effort to develop a special-purpose routine unless the problem were to be solved an extremely large number of times.

In contrast, research concerned with gaining speed in solving large-scale problems must focus on linear algebra issues that may be quite specialized. For large problems, linear algebraic choices are not only more varied than for small problems, but also can exert a dominant influence on solution speed. The tradeoffs are difficult to quantify and analyze, since shortcuts in linear algebra during one part of an algorithm necessarily lead to changes in the algorithm's overall behavior. For example, obtaining an approximate solution to a linear system using a "cheap" method (rather than factorizing a large matrix to solve the system exactly) may increase the number of iterations required to solve the problem, which means that more systems need to be solved. Depending on the increase in the number of iterations and the relative costs per iteration, the cheaper method for solving the system may or may not be worthwhile. An informed decision about the preferred strategy is not straightforward, and is likely to be problem-dependent. (This phenomenon, although unsettling, is familiar in the field of sparse linear algebra.)

Some features of large-scale optimization problems—those that dictate the nature of the linear systems—have an obvious and direct effect on linear algebra. Many large optimization problems arise from modeling of physical systems. When continuous problems are represented in discretized form by piecewise interpolants, for example, the optimal value of any given variable tends to be influenced by only a relatively small number of nearby variables. Such limited and localized interactions often imply sparsity in the resulting Jacobian and Hessian matrices.

More generally, the problem functions may be "partially separable", in the sense that the objective function can be expressed as the sum of element functions on which small subsets of the variables have disjoint effects. For a discussion of the general concept of group-partial separability, see [29]. We note later that partially-separable structure in the problem allows an efficient representation of curvature information; see Section 3.2.

Another feature, less well understood but equally important, of some large problems is that they are highly *structured*—*not* the same as sparse. For example, in a problem involving modeling of closely related physical phenomena, the Jacobian or Hessian may contain different multiples of the same (dense) matrix, or of matrices that differ only in minor ways. Enormous gains in efficiency can result if linear algebraic procedures are designed to exploit this high commonality. For an example of such improvements, see [33].

In addition to sparsity and structure, certain properties of the solution that are usually not known in advance can significantly influence algorithm performance, although they do not appear at first glance relevant to linear algebra. Such properties include: the proportion of nonlinear constraints; the fraction of constraints active at the solution; and the nearness of the constraints to linearity. To illustrate how these affect linear algebra, consider the MINOS code [39], a widely-used software package for large-scale optimization. The algorithm of MINOS explicitly stores and performs rank-one modifications on the Cholesky factor of a dense matrix whose dimension is typically close to the number of degrees of freedom at the solution (roughly speaking, the number of variables minus the number of active constraints). Hence, although MINOS is highly reliable on general problems, it is inherently most effective on problems where the fraction of active constraints is relatively large.

It should be clear that no single algorithm will be "the best" for all large-scale problems. Most optimization algorithms are linked to a (frequently unstated) particular *model of the underlying problem*. Characterization of the underlying problem can be crucial to understanding why certain algorithmic choices are made, such as the form of the linear systems to be solved when several mathematically equivalent alternatives are available. A large-scale algorithm will be most effective when applied to an actual problem that closely matches its underlying problem.

3. Unconstrained Optimization

3.1. Small problems

It has long been considered that Newton's method is the "ideal" optimization method. Newton's method for unconstrained optimization is based on approximating the variation of f near the kth iterate x_k by a local quadratic model derived from the Taylor-series expansion:

$$f(x_k + p) \approx f(x_k) + g_k^T p + \tfrac{1}{2} p^T H_k p, \tag{3.1}$$

where g_k and H_k are the gradient and Hessian evaluated at x_k. Our theme of inexactness makes its first appearance in this most classical of methods, since the quadratic formulation is not in general an exact representation of $f(x_k + p)$.

If H_k is positive definite, the step p_N from x_k to the (unique) minimizer of the local model is called the *Newton direction*, and satisfies the *Newton equations*:

$$H_k p_N = -g_k. \tag{3.2}$$

When $H(x^*)$ is positive definite and x_k is sufficiently close to x^*, a "pure" Newton method in which the next iterate is $x_k + p_N$ displays quadratic convergence to x^* (see, e.g., [9, 10, 15]). Otherwise, Newton's method is reliable only if safeguards are included to ensure convergence from an arbitrary starting point and to treat non-positive-definiteness of the Hessian.

Newton-based methods tend to be categorized as either line search or trust region methods; see [9, 10, 15]. We consider only line search methods; linear algebraic issues analogous to many of those raised for line search methods also apply to trust region methods. In a line search method, the new iterate x_{k+1} satisfies the relation

$$x_{k+1} = x_k + \alpha_k p_k. \tag{3.3}$$

The n-vector p_k, the search direction, is defined using curvature information; its calculation will be discussed later. The scalar α_k is a positive number chosen during a line search to produce a suitable decrease in the value of f compared to that at x_k. A pure Newton method corresponds to $p_k = p_N$ and $\alpha_k = 1$.

Within Newton's method, linear algebra is invariably described as playing the role of "solving the Newton equations". However, this seemingly innocuous statement is vastly oversimplified and can even be misleading. In fact, it may be *undesirable* to solve the Newton equations exactly, for at least three reasons. First, for all problems, small and large, the current local model may be inadequate—for example, the Hessian may be indefinite or singular. With an indefinite Hessian, it is probably undesirable to take a step along the Newton direction; when the Hessian is singular, the Newton direction may not exist. In either case, an alternative procedure is used to define p_k. For large problems, two other factors contribute to the undesirability of solving the Newton equations exactly: the cost of evaluating and storing the exact Hessian may be too high; or the linear algebra needed to solve the Newton equations may be too expensive.

The alert reader should pause at this point to observe an implicit and highly crucial assumption underlying the last two justifications of the preceding paragraph. Asserting that it is "too expensive" to solve the Newton equations, yet assuming that we shall nonetheless succeed in solving the original problem, implies that we do not actually *need* to solve the Newton equations exactly. (This conclusion is so easy for optimizers to accept that it is often taken for granted, thereby leading to confusion.) To belabor the point for emphasis: if solving the Newton equations were essential, failing to solve them would not be acceptable. But, since solving the Newton equations is only a means for reaching the desired end of minimizing f, a cheaper procedure can be used if it will achieve the same result. Consequently the luxury of inexactness, which we shall describe in various forms in the rest of the paper, is permitted as long as it leads to an effective method for solving the given problem.

Although the search direction p_k need not solve the Newton equations exactly, it must satisfy other criteria. In particular, standard line search methods require p_k to be a descent direction for f at x_k, i.e. $p_k^T g_k < 0$. When p_k is a descent direction, then, for all sufficiently small positive steps α, it can be guaranteed that $f(x_k + \alpha p_k) < f(x_k)$. The property that p_k is a descent direction will hold automatically if p_k satisfies the linear system

$$M_k p_k = -g_k, \tag{3.4}$$

for *any* positive definite matrix M_k. We assume henceforth that p_k always satisfies this equation, and much of our discussion will concern the possible choices for M_k.

A second, more elusive, criterion is that p_k should lead to a "good" reduction in f, and to eventual fast convergence with $\alpha_k = 1$ in (3.3). What this means (speaking impressionistically) is that, if p_k satisfies $M_k p_k = -g_k$, then M_k must contain useful curvature information, i.e. information that somehow represents the important properties of the exact Hessian. It is impossible to be precise about this characterization, since the essential Hessian information needed in M_k depends on the problem.

For small problems, a technique widely used to deal with possible indefiniteness or singularity of the Hessian is to apply a *modified Cholesky factorization* to H_k; the two most popular factorizations of this type are described in [24] and [48]. Modified Cholesky factorizations fulfill two purposes simultaneously: they check for positive-definiteness of H_k while calculating the Cholesky factors of a matrix M_k used to define the search direction from (3.4). If H_k is sufficiently positive definite, M_k is taken as H_k, and no modification is required. Otherwise, the Cholesky factors of a *modified* matrix M_k are calculated. The matrix M_k is guaranteed to be positive definite, and is given by $H_k + E_k$, with E_k a non-negative diagonal matrix (*not* a multiple of the identity). The standard modified Cholesky factorizations always exist, and can be shown to satisfy a bound on the size of $\|E_k\|$.

If second derivatives are too expensive but first derivatives are available, *quasi-Newton methods* are both reliable and efficient. These methods, popularized initially in the 1960's, store an *approximation* M_k to the Hessian, calculate p_k from (3.4), and then perform updates of low rank (one or two) to M_k after each iteration. The usual choice for the initial approximation M_0 is the identity, so that the first step is equivalent to steepest descent. Quasi-Newton methods have been shown to have many desirable theoretical and computational properties, including superlinear convergence. With respect to the theme of inexactness, notice that a quasi-Newton matrix M_k is only an approximation to the Hessian. For small problems, a factorization of M_k is typically stored, and the equations (3.4) are solved exactly. Details about quasi-Newton methods may be found in, for example, [9, 10, 15, 24].

Modified Newton and quasi-Newton approaches allow substantial inexactness for small problems. When all goes well, however, the level of inexactness is expected to decrease (or even disappear) as the solution is approached. In a Newton-based method involving a modified Cholesky factorization, the Hessian should not be modified near x^*. With a quasi-Newton method, the changes to M_k should eventually become small in norm as well as in rank, indicating that M_k has become a good representative of the true Hessian.

3.2. Large problems

Once we move to large-scale unconstrained problems, the third issue mentioned previously—the high linear algebra cost of solving the dense Newton equations exactly—becomes significant. In some instances, efficiency can be maintained simply by switching from a dense to a sparse factorization. If exact second derivatives are available and H_k is sparse, its sparse Cholesky factorization can be calculated; see [11]. Of course, care must be taken to perform the modifications needed to treat indefiniteness without impairing efficiency of the sparse factorization; see [48]. When exact second derivatives are expensive but the Hessian has a known sparsity pattern, clever techniques have been developed for choosing vectors that allow a finite-difference approximation to the Hessian to be calculated with a small number of gradient evaluations; see [3].

In some situations, storage limitations or lack of sparsity mean that it is undesirable to factorize the Hessian in any form. If, however, matrix-vector products involving the Hessian can be calculated efficiently, *truncated Newton* methods introduce yet another form of inexactness. The idea is to "solve" the Newton equations inexactly by performing steps of an *iterative method* such as the conjugate-gradient method (customarily with preconditioning), but stopping before the system has been solved to full accuracy. It was shown in [8] that, if the accuracy with which the Newton equations are solved increases appropriately with the iteration number, the desired quadratic convergence rate of Newton's method can be retained. Broadly speaking, quadratic convergence sets in only near the solution, which is thus where the Newton equations need to be solved accurately. At early iterations, a substantial degree of inaccuracy can be tolerated without sacrificing the desired properties of descent and ultimate fast convergence. In an effective truncated Newton method, preconditioned conjugate-gradient iterations are "truncated" (stopped) using a suitable criterion. A preconditioning strategy that accumulates good curvature information is essential to retain fast Newton-like convergence near the solution. Without preconditioning, a large number of conjugate-gradient iterations would be required to solve the Newton equations accurately.

We do not treat the *nonlinear* conjugate-gradient method (see, e.g., [9, 10, 15, 24]), which performs unconstrained minimization by storing only a few n-vectors, because its interesting issues are not primarily linear algebraic.

If calculation of second derivatives is infeasible for a large problem, our discussion of small problems (Section 3.1) suggests that the method of choice would be a quasi-Newton method. For several years in the 1970's, researchers studied sparse quasi-Newton methods in which each quasi-Newton approximation M_k would retain the prescribed sparsity pattern of the true Hessian. Unfortunately, it turned out to be impossible to preserve positive-definiteness of M_k, a specified sparsity pattern, and satisfaction of the standard quasi-Newton condition; see [49]. Hence this idea has been mostly abandoned, at least for the moment, in favor of other closely related but viable alternatives.

When storing an explicit $n \times n$ quasi-Newton approximation is difficult, a *limited-memory* quasi-Newton method can be considered. With these methods, the low-rank updates are represented by storing the n-vectors that define them rather than by performing explicit updates to an approximate Hessian M_k (or its inverse). As usual, the search direction satisfies (3.4), but the updated matrix is not stored or factorized; p_k is actually calculated by performing a sequence of inner products with the saved vectors. Only a small number of the most recent vectors (say, 5 or 10) are stored, and earlier updates are discarded. The theoretical properties of limited-memory methods are not nearly as attractive as those of standard quasi-Newton methods, but these methods have nonetheless been successful in practice on many problems. Recent papers on this topic include, for example, [34, 40].

Although apparently unachievable in general, a quasi-Newton method tailored to structured problems can be realized when f is partially separable. In this case, small, separate, explicit quasi-Newton approximations to "pieces" of the Hessian can be developed and carefully combined to produce a low-storage quasi-Newton approximation M_k containing the most important curvature information. For details, see [28, 29, 7].

In terms of inexactness, note that with a truncated Newton method, a limited-memory quasi-Newton method, or a partially-separable quasi-Newton method, x^* may be found efficiently without ever calculating the exact solution of any system remotely resembling the Newton equations! This observation—that inexactness may be considerable without apparent damage to efficiency—leads to an important point: *many large-scale problems are very easy.* To see how well-behaved they are, consider the famous two-dimensional Rosenbrock's function with its steeply curving valley, which appears in essentially all standard optimization textbooks. An efficient Newton-based method typically requires around 30 iterations to minimize Rosenbrock's function, starting from the usual initial point. In contrast, it is not uncommon for researchers to report successful solution of problems with thousands of variables in 30–50 iterations. The difficulty associated with Rosenbrock's function has obviously not "scaled up" for these large problems, which means that they are special in some way not yet fully understood. (This phenomenon appears to be similar to the fact that real-world linear programs solved by the simplex method never require anything like the known worst-case number of iterations.) Thus, fortunately, the judicious introduction of inexactness appears to be acceptable for many practical large-scale problems.

4. Linear Constraints

We now consider minimizing $f(x)$ subject to a set of linear constraints, possibly including bounds on the variables. Because of space limitations, we mention only a small selection of issues in large-scale linearly constrained optimization—those associated with linear algebra in general, and inexactness in particular. See [15, 24] for further discussion.

4.1. Active-set methods

As in Section 3, we consider line search methods in which the new iterate x_{k+1} is given by $x_k + \alpha_k p_k$. The constraints are presumed to be in the form $Ax \geq b$; equality constraints and bounds can be handled in a conceptually straightforward way.

Our attention in this paper is limited to the class of *active-set methods*. Active-set methods have been criticized in recent years for their inefficiency, mostly for the "classical" or "one-at-a-time" strategy used to modify the working set, which we describe later. Various alternatives have been proposed, primarily for the case of convex quadratic programming with bound constraints; see, for example, [38]. Despite some developments in non-active-set software, no clear "winner" has emerged for practical solution of general linearly constrained problems; furthermore, active-set methods provide a useful paradigm for understanding and developing new algorithms. The linear algebra issues associated with active-set methods are sufficiently close to those for newer strategies so that this discussion is likely to remain relevant even if active-set methods are superseded.

The motivation for active-set methods derives from the following observations. Suppose that the set of constraints *active* (i.e., that hold with equality) at x^* is known; let $\hat{m}$ denote the number of active constraints, $\hat{A}$ the Jacobian of the active constraints, and $\hat{b}$ the associated subvector of b. (We assume for simplicity that $\hat{A}$ has full row rank.) Finally, assume that the current iterate x_k is feasible and satisfies $\hat{A}x_k = \hat{b}$. Let Z denote a basis for the null space of $\hat{A}$, so that $\hat{A}Z = 0$, and define p_k as Zp_Z for some $(n - \hat{m})$-vector p_Z. Then

$$\hat{A}(x_k + \alpha p_k) = \hat{A}x_k + \alpha \hat{A} Z p_Z = \hat{b}.$$

By moving only in the null space of $\hat{A}$, subsequent iterates will automatically stay "on" the active constraints.

If, as in the unconstrained case, a local quadratic model is defined of the variation in f (see (3.1)), but restricted to directions in the null space of $\hat{A}$, we have

$$f(x_k + Zp_Z) \approx f_k + p_Z^T Z^T g_k + \tfrac{1}{2} p_Z^T Z^T H_k Z p_Z. \tag{4.1}$$

The number of degrees of freedom in (4.1) is the dimension of p_Z, and we may apply unconstrained techniques within a reduced subspace to choose a "good" p_Z. The vector $Z^T g_k$ and the matrix $Z^T H_k Z$ are called the reduced gradient and reduced Hessian. If p_Z satisfies the *reduced Newton equations*:

$$Z^T H_k Z p_Z = -Z^T g, \tag{4.2}$$

then $p_k = Zp_Z$ is the exact analogue of the Newton direction p_N for an unconstrained problem; compare (3.2).

In general, of course, the true active set is *unknown*, and must be predicted. An active-set method is based on maintaining a prediction of the active set (called the *working set*) that is used to define the search direction. At any given iteration, constraints in the working set are treated as equalities for the purpose of defining Z and the reduced gradient/Hessian.

Assume that the Jacobian of constraints in the current working set is $\hat{A}_k$, and that $\hat{A}_k Z = 0$. When $Z^T g_k \neq 0$, the search direction is defined as Zp_Z, where p_Z is chosen so that p_k is a descent direction for f; for instance, p_Z could solve some version of the reduced Newton equations (4.2). After a suitable p_k is determined, a steplength $\alpha_k > 0$ is chosen such that $x_k + \alpha_k p_k$ is feasible and $f(x_k + \alpha_k p_k)$ is sufficiently less than $f(x)$. If the step α_k "hits" a constraint not currently in the working set, this constraint is added to the working set before beginning the next iteration.

If $Z^T g_k = 0$, tests are performed to see whether x_k is optimal. Such tests involve the signs of the Lagrange multipliers with respect to the working set, and the inertia of the current reduced Hessian. If the reduced Hessian is positive definite and a Lagrange multiplier is negative, the associated constraint is deleted from the working set. Other, more complicated, situations arise that will not be considered here. Note that, according to this description, constraints are added to or deleted from the working set one by one, so that each change in the working set is a *rank-one modification* to $\hat{A}_k$ (adding or deleting a row). This strategy for altering the working set is *not* essential for an active-set method (in fact, other approaches have been suggested), but its abandonment leads to more complicated linear algebra, especially for general linear constraints.

4.2. Defining a basis for the null space

Based on the reduced Newton equations (4.2) and the required form Zp_Z for p_k, a straightforward transcription of an active-set method would involve calculation of a null-space basis Z. For small problems, an explicit Z is typically obtained by processing the orthogonal transformations used to compute a QR factorization of $\hat{A}_k$; this produces an orthonormal basis for the null space, which has several numerical advantages. The linear algebra involved in a dense active-set method might thus include updating the QR factorization as $\hat{A}$ changes, and updating the chosen representation of the reduced Hessian.

For large problems in which the Jacobian of the constraints is sparse, techniques for computing and updating an orthonormal null-space basis have been studied by several researchers (see, e.g., [4]). Unfortunately, such calculations remain relatively expensive, and research is still proceeding. A popular and long-standing alternative is to use the "reduced-gradient form of Z", which is closely related to procedures associated with the simplex method for standard-form linear programming. If the working set matrix $\hat{A}_k$ is expressed as $(B \;\; S)$, where B is square and nonsingular, the columns of the following matrix Z constitute a (generally non-orthonormal) basis for the null space of $\hat{A}_k$:

$$Z = \begin{pmatrix} -B^{-1}S \\ I \end{pmatrix}. \tag{4.3}$$

The reduced-gradient Z is *not* formed explicitly. Rather, products of Z and Z^T with vectors are computed by solving linear systems involving B and B^T. These systems are typically solved using a sparse LU factorization of B, which is updated as $\hat{A}_k$ changes.

The reduced-gradient form of Z is highly efficient for many large problems, and features in, for example, the MINOS code [39]. However, it is possible to remain in the null space without explicit reference to Z; a technique for doing so is considered in Section 4.4.

4.3. The reduced Newton equations

In addition to ensuring that p_k lies in the null space of $\hat{A}$, an effective active-set method must somehow define a "good" direction p_Z. By direct analogy with the unconstrained case, p_Z might be computed by solving the reduced Newton equations (4.2) or a quasi-Newton version thereof, or by inexactly solving such a system.

A serious difficulty is that, despite availability and sparsity of H_k, the reduced Hessian $Z^T H_k Z$ is not necessarily sparse. Furthermore, calculating the reduced Hessian from scratch can be quite expensive, even with the reduced-gradient form of Z (4.3). Several options have been proposed when the exact Hessian is available, the most obvious of which involve application of a truncated Newton method; an open question is the choice of preconditioner.

If second derivatives are expensive and the dimension of Z is reasonably small, a quasi-Newton approximation of the *reduced Hessian* can be maintained, as in MINOS [39]. If the working set remains fixed during a given iteration, the approximate reduced Hessian can be updated as usual. If the working set is altered, Z changes dimension, and the effect of a modified Z must somehow be reflected in the reduced Hessian approximation. An open research area today is the issue of representing and updating an approximate reduced Hessian when f is partially separable or group-partially separable.

In the formulation described thus far of an active-set method for linear constraints, inexactness plays two different roles. Although it is clearly possible, perhaps even desirable, to solve the reduced Newton equations inexactly, it is *unacceptable* to use an inexact Z. With linear constraints, an active-set method depends on staying in the null space, which means that equations enforcing this property must be solved accurately. To paraphrase George Orwell's *Animal Farm*, "Some equations are more equal than others".

4.4. The Karush-Kuhn-Tucker system

We now consider whether the search direction can be forced to remain in the null space without explicitly invoking a null-space basis Z. An increasingly popular approach, especially well suited for sparse problems, is to calculate p_k by solving a larger system whose result is mathematically equivalent to the desired effect. Although this may seem contradictory, it can be faster to factorize a sparse but much larger system (see (4.5), below) from scratch than to perform a single quasi-Newton update on a dense (albeit reduced) Hessian.

The vector p_k satisfying

$$p_k = Zp_Z, \quad \text{with} \quad Z^T H_k Z p_Z = -Z^T g_k, \tag{4.4}$$

is also part of the solution of the $(n + \hat{m})$-dimensional symmetric indefinite *Karush-Kuhn-Tucker* (KKT) equations:

$$\begin{pmatrix} H_k & \hat{A}_k{}^T \\ \hat{A}_k & 0 \end{pmatrix} \begin{pmatrix} p_k \\ -\lambda_k \end{pmatrix} = K \begin{pmatrix} p_k \\ -\lambda_k \end{pmatrix} = \begin{pmatrix} -g_k \\ 0 \end{pmatrix}. \tag{4.5}$$

Because of the last $\hat{m}$ equations, the vector p_k must satisfy $\hat{A}_k p_k = 0$, and hence lies by definition in the null space of $\hat{A}_k$. With exact arithmetic, solving the KKT system produces the identical direction as the formulation (4.4) in which Z appears.

The eigenvalue structure (inertia) of K is closely related to that of the reduced Hessian; see, for example, [23]. In particular, when $\hat{A}_k$ has full row rank and the reduced Hessian is positive definite, K is nonsingular. We shall see later that the KKT system plays a significant role in SQP methods for nonlinearly constrained optimization and in interior methods; see Section 5.1.

For problems in which the Hessian H_k and Jacobian $\hat{A}_k$ are sparse, the KKT system would most logically be solved by calculating a sparse symmetric indefinite factorization of K; see [11] for a discussion of sparse factorization techniques. When standard software of this kind was first applied extensively to KKT systems (in the mid- to late-1980's), a loss of efficiency was observed compared to its usual performance. To explain this phenomenon, we note that the software conducts an initial "analyze" phase devoted to constructing a reordering based on the sparsity pattern, and then executes a numerical "factorize" phase in which interchanges are performed as necessary for stability. Because most matrices to which the software had been applied arose from the context of partial differential equations, the analyze phase was based on the presumption that the diagonal elements were non-zero (which is clearly not true for KKT systems); because of this assumption, many more interchanges than expected were necessitated in the factorization phase.

Since then, new codes have been developed that do not suffer from this difficulty. In [12], a general sparse symmetric indefinite factorization code is presented that uses several new strategies for promoting sparsity and retaining stability. A sparse indefinite solver developed for interior methods (see Section 5.2) is described in [17] that guards against poor orderings by combining the analysis and factorization phases. A more specialized strategy is based on the observation that the sparsity patterns of H_k and $\hat{A}_k$ in the KKT system are related in certain large-scale problems, and it may be possible to calculate the factorization more efficiently by exploiting this special structure; see, e.g., [22].

The solution of KKT systems is an example where an application driven by optimization has led to a change in strategy in a popular "black-box" linear algebra package. Several interesting research questions remain open concerning the KKT approach. One of the most pressing involves development of reliable and effective strategies for coping with an "inappropriate" Hessian, which in this context is the reduced Hessian. The basic difficulties of detecting and treating indefiniteness and singularity are the same as those mentioned in our earlier discussion of unconstrained optimization (Section 3.1), greatly exacerbated because the relevant matrix (the reduced Hessian) is *not* available. The linear system (4.5) being solved does not directly involve the reduced Hessian; rather, the full Hessian H_k is embedded in the larger KKT matrix. The best way to detect and modify indefiniteness or singularity of the reduced Hessian while factorizing K is unclear, as is the most effective strategy for ensuring that p_k will be a descent direction for f. See, for example, [16].

As with solving the reduced Newton equations (4.2), it is tempting to suggest that the KKT system should be solved iteratively, and perhaps inexactly. Since the KKT matrix is not positive definite, a special form of the conjugate-gradient method must be used, such as SYMMLQ [43]. Specialized preconditioners for KKT systems are also advisable; see [18]. If a substantial level of inexactness is allowed, the crucial relation $\hat{A}_k p_k = 0$ that keeps p_k in the null space may not be satisfied. Whether this is acceptable depends on the situation. In some instances, remaining in the null space is essential; in others, however, a certain level of inaccuracy is tolerable, such as when KKT systems are solved as subproblems within interior methods for nonlinearly constrained problems (see Section 5.2).

4.5. Quadratic programming

Much attention has been devoted to active-set methods for linear and quadratic programming. (In fact, the simplex method for linear programming is the most famous active-set method.) Rather than attempt to cover linear programming, we refer the reader to the recent survey [25], which discusses both simplex and interior methods.

The literature on dense quadratic programming (QP) is vast; see, for example, [15, 23]. Both [22] and [27] describe large-scale QP methods based on solving the KKT system (4.5), with highly specialized techniques that exploit the quadratic nature of the objective function. In the Schur-complement QP method [22], a sparse factorization is calculated of the KKT matrix defined with a suitable initial working set $\hat{A}_0$. Although change in the working set leads to a theoretically altered version of K, the needed quantities can be calculated using the initial (fixed) factorization and a dense factorization of the Schur complement, a matrix whose dimension is equal to the number of QP iterations. If the number of QP iterations is reasonably small—say, in the hundreds—this technique can be quite efficient.

One reason for interest in quadratic programming is that QP subproblems occur in the extremely effective class of sequential quadratic programming (SQP) methods for nonlinearly constrained optimization (see Section 5.1). Progress in large-scale quadratic programming is therefore likely to have a major effect on large-scale nonlinearly constrained problems as well.

5. Nonlinear Constraints

Consider the generic nonlinear inequality-constrained problem

$$\underset{x \in \mathcal{R}^n}{\text{minimize}}\ f(x) \quad \text{subject to} \quad c_j(x) \geq 0, \;\; j = 1, \ldots, m, \tag{5.1}$$

where for simplicity of exposition we include only inequality constraints. We have omitted detailed consideration of nonlinear equality constraints because of space limitations and because their treatment is less focused on linear algebra, *not* because they are "easy".

The terminology in discussing (5.1) is slightly more complicated than for the linearly constrained case. Assume that $\hat{m}$ constraints are active at the solution x^*; for any point x, $\hat{A}(x)$ means the $\hat{m} \times n$ Jacobian of the constraints active at x^*, evaluated at x. Let $Z(x)$ denote a matrix whose columns form a basis for the null space of $\hat{A}(x)$, so that $\hat{A}(x)Z(x) = 0$. In general $Z(x)$ is not unique, but may be defined as a smooth function in regions where $\hat{A}(x)$ has full rank (see [5]). Let Z^* denote a basis for the null space of $\hat{A}(x^*)$, and Y^* a basis for the range space of $\hat{A}^T(x^*)$.

To simplify our discussion, it is convenient to assume that the following conditions (second-order sufficiency as well as strict complementarity) hold at x^*:

(i) $g(x^*) = A^T(x^*)\lambda^*$, where λ^* is called the Lagrange multiplier vector;

(ii) $\lambda_j^* c_j(x^*) = 0$ for $j = 1, \ldots, m$;

(iii) $\lambda_j^* > 0$ if constraint j is active;

(iv) the gradients of the active constraints at x^* are linearly independent, i.e. $\hat{A}(x^*)$ has full row rank;

(v) $Z^{*T} W^* Z^*$ is positive definite, where $W^* = H(x^*) - \sum_{j=1}^m \lambda_j^* H_j(x^*)$; the matrix W^* is the Hessian of the Lagrangian function evaluated at x^*.

For large-scale problems with nonlinear constraints, the most popular solution technique for many years has been the method of MINOS [39]. More recently, the code LANCELOT [7], which is based on trust-region and augmented Lagrangian methods, has become widely available. Both of these provide reliable and soundly-based software for general large-scale problems.

In this section, we treat two other approaches that are less well established for large problems, but involve some interesting linear algebra issues.

5.1. Sequential quadratic programming (SQP) methods

For small nonlinearly constrained problems with inequality constraints, arguably the most popular technique for at least ten years has been that of sequential quadratic programming (SQP); detailed discussions of SQP methods are given in, for example, [45, 15, 24]. Several suggestions have been made for applying trust-region techniques to nonlinear constraints, but most of these thus far have been limited to equality constraints; see [2, 50].

A line search SQP method (the only kind to be considered here) chooses the search direction p_k as the solution of a *quadratic programming subproblem* in which the quadratic objective represents the local variation in the *Lagrangian function*. As in (3.3), the new iterate is taken as $x_k + \alpha_k p_k$, where the step α_k is usually chosen to yield a sufficient decrease in a merit function (a suitable combination of the objective and constraint functions).

In developing SQP methods, a range of strategies has been implemented to balance the gain in speed from a correct early identification of the active set against the loss of efficiency caused by an incorrect prediction. By analogy with the linearly constrained case, a so-called "EQP" strategy can be applied to define a QP subproblem containing only equality constraints corresponding to linearizations of constraints in a specified working set (a prediction of constraints active at the solution). The associated QP subproblem is

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}p^T W_k p + p^T g_k \\ \text{subject to} & \hat{A}_k p = -\hat{c}_k, \end{array} \tag{5.2}$$

where W_k is an approximation of the Hessian of the Lagrangian, g_k is the gradient of f, $\hat{A}_k$ is the Jacobian of constraints in the working set at x_k, and $\hat{c}_k$ is the vector of constraints in the working set. Let Y be a basis for the range space of $\hat{A}_k{}^T$, and Z a basis for the null space of $\hat{A}_k$.

The solution of (5.2) can be expressed as

$$p = Yp_Y + Zp_Z, \tag{5.3}$$

where p_Y and p_Z satisfy the following equations:

$$\hat{A}_k Y p_Y = -\hat{c}_k; \tag{5.4}$$

$$Z^T W_k Z p_Z = -Z^T g_k. \tag{5.5}$$

The benefits of an EQP approach mainly involve simplification in the computational and informational requirements for calculating p_k. In particular, no iterations are required to solve the QP subproblem (5.2), and only the reduced Hessian of the Lagrangian function appears in (5.5). A distinction from the equations (4.2) for the linearly constrained case is that, if $\hat{c}_k \neq 0$, p_k has a non-zero portion in the range space of $\hat{A}_k{}^T$ (see (5.4)), and hence a basis Y for this subspace enters the calculations. The major difficulty with an EQP approach is that a poor choice of working set may lead to inefficiency; unfortunately, iron-clad strategies for predicting the active set do not exist.

At the other extreme is a philosophy that, in effect, lets the QP subproblem determine its own active set, and includes linearizations of *all* the constraints as inequalities:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}p^T W_k p + p^T g_k \\ \text{subject to} & A_k p \geq -c_k, \end{array} \tag{5.6}$$

where A_k is the complete Jacobian and c_k is the vector of constraint values. A full-blown "IQP" strategy assumes exact solution of (5.6), which not only involves an unpredictable number of QP iterations, but also requires a representation of the complete matrix W_k rather than the reduced matrix $Z^T W_k Z$. On a typical nonlinearly constrained problem, solving (5.6) is more costly than solving (5.2), but the total number of "outer" iterations (QP subproblems) needed to solve the original problem is smaller. It is reasonably well established that neither strategy is consistently better on all problems, and many researchers have sought a compromise between the two.

Techniques based on availability of only a reduced Hessian have been examined for several years; see [42, 30]. In effect, the reduced Hessian is "unprojected" back into the full space so that the QP iterations can proceed; however, the solution obtained in this way may be the exact solution of a QP different from (5.6). The recent algorithm in [13] is specifically designed for large sparse problems, and uses a combination of dense and sparse techniques, including special sparse forms for the needed subspace bases.

A somewhat different perspective for small problems is applied in [46], which begins with the subproblem (5.6), but does not necessarily solve it to completion. This strategy provides yet another variant on the concept of an "inexact" solution. We stress that "partially solving" (5.6) by performing only a few iterations of a standard active-set QP method does not ensure that the solution so found is close in any sense to the exact solution. On the other hand, the iterations performed generate information that may be used to construct a descent direction for a suitable SQP merit function.

Whatever the choice of QP subproblem, all SQP algorithms "solve" (in some sense) KKT systems of the form (4.5), or a sequence of such systems. In some cases the (2, 2) block may be nonzero; for instance, the Sℓ_2QP method [45] involves solving a KKT-like system in which the $(2,2)$ block is a diagonal matrix converging to zero.

Substantial progress in the development of large-scale SQP methods has already occurred, and is likely to continue, perhaps with a proliferation of mixed techniques. For example, finite-differencing can be used to approximate products of the Hessian with selected vectors, such as columns of Z, the null-space basis; this highly effective technique has been applied within a combined SQP-interior method (see [47]). In [1], an SQP method is described in which the constraint Jacobian is stored in sparse form, the subproblems are solved using a Schur-complement QP method (see Section 4.5), and the QP Hessian is represented by limited-memory quasi-Newton updates (see Section 3.2), modified by a trust-region strategy to ensure positive-definiteness. Second-derivative SQP methods are likely to be especially effective on sparse problems once QP-related issues involving the proper treatment of indefiniteness and singularity are resolved; see Section 4.4.

5.2. Interior methods

In 1984, Karmarkar announced his discovery of a polynomial-time method for linear programming [32], along with numerical tests showing that it out-performed the simplex method on large problems. In [20], this method was shown to have a formal connection with classical barrier methods for nonlinear programming. Since then, enormous activity on barrier and other interior methods has continued, focused mainly on complexity rather than numerical results. The classic reference on interior methods is [14]; more recent surveys include [26] and [52].

Interior methods deal with inequality constraints and, in their original form, construct a sequence of strictly feasible iterates (hence the designation "interior"). Given the inequality-constrained problem (5.1), the associated logarithmic barrier function is

$$B(x,\mu) = f(x) - \mu \sum_{i=1}^{m} \log c_i(x), \tag{5.7}$$

where the positive parameter μ is called the *barrier parameter*. Let x_μ denote an unconstrained minimizer of $B(x,\mu)$. Under reasonably mild assumptions, it can be shown that $\lim_{\mu\to 0} x_\mu = x^*$. Here, for simplicity of exposition we have assumed second-order sufficiency and strict complementarity. Under these added conditions, the sequence $\{x_\mu\}$ defines a differentiable trajectory converging to x^* nontangentially to the active constraint normals. For details, see, e.g., [52].

It would appear that using an interior method to solve a constrained problem simply involves solving a sequence of unconstrained problems, namely minimizing $B(x,\mu)$ for a suitable selection of barrier parameters converging to zero. This was indeed a large part of the motivation for barrier methods in the 1960's, when techniques for unconstrained optimization were far more developed than those for constrained problems. However, barrier methods fell from favor for a variety of reasons, and were in fact dismissed as outmoded for many years. One reason for their loss of popularity was a perceived (and observed) inefficiency compared with alternative methods, such as augmented Lagrangian methods (on which LANCELOT [7] is based) and SQP methods. Another reason involves linear algebra, which we now discuss in some detail.

The barrier Hessian with respect to x, denoted by H_B, is given by

$$H_B(x,\mu) = H(x) - \sum_{j=1}^{m} \frac{\mu}{c_j(x)} H_j(x) + \mu A^T(x) C^{-2}(x) A(x), \tag{5.8}$$

where $C^2(x)$ is the matrix $\operatorname{diag}(c_j^2(x))$. We define the m-vector $d(x,\mu)$ as

$$d_j(x,\mu) = \frac{\mu}{c_j(x)}, \quad j = 1,\dots,m. \tag{5.9}$$

It follows from our assumptions of second-order sufficiency and strict complementarity that, at points x_μ on the barrier trajectory, the value of d_j is within $O(\mu)$ of λ_j^*, the optimal Lagrange multiplier; see [14, 52].

(In stating results about the barrier Hessian, we use the standard notation $O(\cdot)$, $\Omega(\cdot)$ and $\Theta(\cdot)$ from complexity theory; see, for example, [44]. If ϕ is a function of a positive variable h, p is fixed, and there exists a constant $\kappa_u > 0$ such that $|\phi| \le \kappa_u h^p$ for all sufficiently small h, then $\phi = O(h^p)$. If there exists a constant $\kappa_l > 0$ such that $|\phi| \ge \kappa_l h^p$ for all sufficiently small h, then $\phi = \Omega(h^p)$. If there exist constants $\kappa_l > 0$ and $\kappa_u > 0$ such that $\kappa_l h^p \le |\phi| \le \kappa_u h^p$ for all sufficiently small h, then $\phi = \Theta(h^p)$.)

It can be shown that, near x^*, the barrier Hessian exhibits a special structure in its eigenvalues and invariant subspaces. In particular, except in special cases the barrier Hessian is ill-conditioned in an entire neighborhood of x^*. It is shown in [53] that, for all points sufficiently near x^*, under appropriate assumptions,

$$\mathrm{cond}(\mu H_B) = \mathrm{cond}(H_B) = d_{\max}\, \Omega(1/\mu),$$

where $d_{\max}$ is the maximum element of the vector d of (5.9). The value of $d_{\max}$ can be unboundedly large if we permit points very close to the boundary of the feasible region, for which a constraint is inordinately small relative to μ. For points close to x^* with the additional property that c_j is $\Theta(\mu)$ for all j, so that $d_{\max} = \Theta(1)$, then the barrier Hessian H_B has simple invariant subspaces that are "close to" (within $O(\mu)$ of) those defined by Z^* and Y^*. For a detailed discussion, see [53].

These results can be interpreted as a statement that, as the iterates of a barrier method converge, the invariant subspace associated with the unbounded ("large") eigenvalues of the barrier Hessian is close to the range space of $\hat{A}^T$, and the invariant subspace associated with the bounded ("small") eigenvalues is close to the null space of $\hat{A}$. For the scaled matrix μH_B, one might say informally that its "small space" (the subspace of vectors v for which $\|\mu H_B v\|$ is small relative to $\|v\|$), is close to the null space of $\hat{A}$.

Because the performance of Newton-like methods is affected adversely by ill-conditioning in the Hessian, this feature of the barrier function causes difficulties for "black-box" unconstrained methods. Fortunately, the structure of the ill-conditioning is known, and need not be a fatal flaw if appropriate care is taken. In fact, the analysis sketched above shows that barrier Hessian itself reveals approximate subspace information about the active constraint Jacobian at x^*.

The saving grace for barrier methods is that, at points near the barrier trajectory, a highly accurate approximation to the step to x^* is given by $p = Yp_Y + Zp_Z$, where Y and Z are bases for range and null spaces associated with the current active constraint Jacobian, and p_Y and p_Z satisfy

$$\hat{A}Yp_Y = v_1, \qquad Z^TWZp_Z = Z^Tv_2, \tag{5.10}$$

for vectors v_1 and v_2 closely related to those of (5.4) and (5.5). The crucial point is that the condition of both linear systems in (5.10) reflects that of the problem (5.1) itself, and is not large simply as an artifact of applying a barrier transformation. The ability to approximate a step to x^* while solving well-conditioned systems motivated the barrier trajectory algorithm of [51], and has appeared more recently in the algorithm proposed in [41].

Unfortunately, approaches based on solving the equations (5.10) must make an estimate of the unknown active set $\hat{A}$. Although an incorrect prediction of $\hat{A}$ in this context has only a local effect on the algorithm, an important feature of interior methods is that they do not make an explicit prediction of the active set. To resolve this apparent dilemma, we need to ask whether there are linear algebraic techniques that allow us to cope with asymptotic ill-conditioning whose structure is known in advance, but which is revealed only implicitly. Note that the barrier Hessian will in general remain nonsingular even close to x^*; it is singular only in the limit.

For small problems, a strategy has been developed [53] based on the rank-revealing Cholesky factorization, which is analyzed in detail in [31]. While factorizing the barrier Hessian H_B, a decision is made about its nearness to singularity. If H_B is safely nonsingular, the Cholesky factors can be used in the standard way to calculate a Newton-based direction; see Section 3.1. If H_B is deemed numerically rank-deficient, it is "replaced" by an exactly rank-deficient matrix constructed from the Cholesky factors, which can be transformed to provide approximate bases Y and Z for the needed range and null spaces. In complete contrast to the case of linear constraints, here Y and Z *are* permitted to be inexact, since it is not necessary (or even possible) to remain in the null space. Near x^*, when the inevitable ill-conditioning strikes, approximate subspace information can thus be deduced that allows calculation of a good approximation to the Newton direction, without solving the ill-conditioned Newton equations and without explicitly predicting the active set. Inexactness cannot be too extreme, since the decision about rank of the barrier Hessian must be correct to obtain the desired Newton-like behavior near x^*. In any case, a satisfactory descent direction for the barrier function can be calculated.

The work of [41], an interior method for large-scale problems with bound constraints, calculates the search direction in separate range- and null-space components based on a working set, and solves the Newton-like equations in (5.10) inexactly using a preconditioned truncated conjugate-gradient method. For *general* large-scale nonlinearly constrained problems, however, serious application of interior methods is not particularly advanced.

An approach suggested in [19] in the context of linear and quadratic programming is to solve a larger, KKT-like system in which the known ill-conditioning can be dealt with more directly. Linear systems involving H_B can be solved via systems containing the larger symmetric matrices

$$\begin{pmatrix} -C_k^2/\mu & A_k \\ A_k^T & W_k \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \mu C_k^{-2} & 0 & I \\ 0 & W_k & A_k^T \\ I & A_k & 0 \end{pmatrix}, \tag{5.11}$$

where C_k is the diagonal matrix of constraint values at x_k. The ill-conditioning in these matrices is revealed in the $(1,1)$ block, and may be treated by a special interchange strategy for calculating the symmetric indefinite factorizations.

Another highly promising approach, which remains to be explored and implemented, is the primal-dual technique proposed in [35], which avoids ill-conditioning by abandoning symmetry. In this method, the systems to be solved contain matrices of the form

$$\begin{pmatrix} W_k & A_k^T \\ \Lambda_k A_k & -C_k \end{pmatrix}, \tag{5.12}$$

where Λ_k is a diagonal matrix of Lagrange multiplier estimates for *all* the constraints, including active and inactive. Because of strict complementarity (relation (iii) at the beginning of this section), the constraints converging to zero correspond to strictly positive Lagrange multipliers, and inactive constraints correspond to Lagrange multipliers converging to zero. The matrix (5.12) does *not* become ill-conditioned as x^* is approached, which is why its application might be effective. (There is, of course, an obvious relationship (via an ill-conditioned transformation) between (5.12) and the matrices of (5.11).)

6. Conclusions

Linear algebra in large-scale optimization comes in many flavors, ranging from the simplest "vanilla" to a complicated mixture of ingredients involving low-rank updates, representation of matrices in compact form, and special techniques for detecting and treating ill-conditioning. Despite enormous progress, many large optimization problems cannot be solved satisfactorily today. Happily for researchers in linear algebra and optimization as well as practitioners with real problems to solve, these difficulties are being ameliorated by continuing improvements in both fields.

References

[1] J. T. Betts and P. D. Frank, A sparse nonlinear optimization algorithm, Report AMS-TR-173, Boeing Computer Services, Seattle, Washington, 1991.

[2] R. H. Byrd, R. A. Tapia and Y. Zhang, An SQP augmented Lagrangian BFGS algorithm for constrained optimization, *SIAM J. Optim.* 2, 210–241, 1992.

[3] T. F. Coleman and J. J. Moré, Estimation of sparse Hessian matrices and graph coloring problems, *SIAM J. Numer. Anal.* 28, 1984.

[4] T. F. Coleman and A. Pothen, The null space problem I. Complexity, *SIAM J. Alg. Disc. Meth.* 7, 527–537, 1986.

[5] T. F. Coleman and D. C. Sorensen, A note on the computation of an orthogonal basis for the null space of a matrix, *Math. Prog.* 29, 234–242, 1984.

[6] A. R. Conn, N. Gould and P. L. Toint, Large-scale nonlinear constrained optimization, Report 92-02, Facultés Universitaires de Namur, Namur, Belgium, 1992.

[7] A. R. Conn, N. Gould and P. L. Toint, LANCELOT*: a Fortran package for large-scale nonlinear optimization*, Springer-Verlag, Berlin and New York, 1992.

[8] R. S. Dembo, S. C. Eisenstat and T. Steihaug, Inexact Newton methods, *SIAM J. Numer. Anal.* 19, 400–408, 1982.

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

[10] J. E. Dennis, Jr. and R. B. Schnabel, A view of unconstrained optimization, in *Optimization* (G. L. Nemhauser, A. H. G. Rinnooy Kan and M. J. Todd, eds.), Elsevier Science Publishers, North-Holland, Amsterdam, 1–72, 1989.

[11] I. S. Duff, A. M. Erisman and J. K. Reid, *Direct methods for sparse matrices*, Oxford University Press, London, 1986.

[12] I. S. Duff, N. I. M. Gould, J. K. Reid, J. A. Scott and K. Turner, The factorization of sparse symmetric indefinite matrices, Report RAL-90-066, Rutherford Appleton Laboratory, Oxon, United Kingdom, 1990.

[13] S. K. Eldersveld, Large-scale sequential quadratic programming algorithms, Report SOL 92-02, Department of Operations Research, Stanford University, Stanford, California, 1992.

[14] A. V. Fiacco and G. P. McCormick, *Nonlinear programming: sequential unconstrained minimization techniques*, John Wiley and Sons, New York, 1968. Republished by SIAM, Philadelphia, 1990.

[15] R. Fletcher, *Practical methods of optimization* (second edition), John Wiley and Sons, Chichester and New York, 1987.

[16] A. L. Forsgren and W. Murray, Newton methods for large-scale linear equality-constrained minimization, Report SOL 90-6, Department of Operations Research, Stanford University, Stanford, California, 1990.

[17] R. Fourer and S. Mehrotra, Performance of an augmented system approach for solving least-squares problems in an interior-point method for linear programming, *Committee on Algorithms Newsletter* 19, 26–31, 1991.

[18] P. E. Gill, W. Murray, D. B. Ponceleón, and M. A. Saunders, Preconditioners for indefinite systems arising in optimization, *SIAM J. Matrix Anal. Appl.* 13, 292–311, 1992.

[19] P. E. Gill, W. Murray, D. B. Ponceleón, and M. A. Saunders, Solving reduced KKT systems in barrier methods for linear and quadratic programming, Report SOL 91-7, Department of Operations Research, Stanford University, Stanford, California, 1991.

[20] P. E. Gill, W. Murray, M. A. Saunders, J. A. Tomlin and M. H. Wright, On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method, *Math. Prog.* 36, 183–209, 1986.

[21] P. E. Gill, W. Murray, M. A. Saunders and M. H. Wright, Constrained nonlinear programming, in *Optimization* (G. L. Nemhauser, A. H. G. Rinnooy Kan and M. J. Todd, eds.), Elsevier Science Publishers, North-Holland, Amsterdam, 171–210, 1989.

[22] P. E. Gill, W. Murray, M. A. Saunders and M. H. Wright, A Schur-complement method for sparse quadratic programming, in *Reliable numerical computation* (M. G. Cox and S. J. Hammarling, eds.), Clarendon Press, Oxford, 113–138, 1990.

[23] P. E. Gill, W. Murray, M. A. Saunders and M. H. Wright, Inertia-controlling methods for general quadratic programming, *SIAM Review* 33, 1–36, 1991.

[24] P. E. Gill, W. Murray and M. H. Wright, *Practical optimization*, Academic Press, London, 1981.

[25] D. Goldfarb and M. J. Todd, Linear programming, in *Optimization* (G. L. Nemhauser, A. H. G. Rinnooy Kan and M. J. Todd, eds.), Elsevier Science Publishers, North-Holland, Amsterdam, 73–170, 1989.

[26] C. C. Gonzaga, Path-following methods for linear programming, *SIAM Review* 34, 167–224, 1992.

[27] N. I. M. Gould, An algorithm for large-scale quadratic programming, *IMA J. Numer. Anal.* 11, 299–324, 1991.

[28] A. Griewank and P. L. Toint, Partitioned variable metric updates for large sparse optimization problems, *Numer. Math.* 39, 119–137, 1982.

[29] A. Griewank and P. L. Toint, Numerical experiments with partially separable optimization problems, in *Numerical analysis: proceedings Dundee 1983* (D. F. Griffiths, ed.), Lecture notes in mathematics 1066, Springer-Verlag, Berlin, 203–220, 1984.

[30] C. B. Gurwitz and M. L. Overton, SQP methods based on approximating a projected Hessian matrix, *SIAM J. Sci. Stat. Comp.* 10, 631–653, 1989.

[31] N. J. Higham, Analysis of the Cholesky decomposition of a semi-definite matrix, in *Reliable numerical computation* (M. G. Cox and S. J. Hammarling, eds.), Clarendon Press, Oxford, 161–185, 1990.

[32] N. K. Karmarkar, A new polynomial time algorithm for linear programming, *Combinatorica* 4, 373–395, 1984.

[33] L. Kaufman and G. Sylvester, Separable nonlinear least-squares with multiple right-hand sides, *SIAM J. Matrix Anal. Appl.* 13, 68–89, 1992.

[34] D. C. Liu and J. Nocedal, On the limited memory BFGS method for large scale optimization, *Math. Prog.* 45, 503–528, 1989.

[35] G. P. McCormick, The superlinear convergence of a nonlinear primal-dual algorithm, Report T-550/91, Department of Operations Research, George Washington University, Washington, DC, 1991.

[36] J. J. Moré, Recent developments in algorithms and software for trust region methods, in *Mathematical programming: the state of the art* (A. Bachem, M. Grötschel and B. Korte, eds.), Springer-Verlag, Berlin and New York, 258–287, 1983.

[37] J. J. Moré and D. C. Sorensen, Newton's method, in *Studies in numerical analysis* (G. H. Golub, ed.), Mathematical Association of America, Providence, Rhode Island, 29–82, 1984.

[38] J. J. Moré and G. Toraldo, Algorithms for bound-constrained quadratic programming, *Numer. Math.* 55, 377-400, 1989.

[39] B. A. Murtagh and M. A. Saunders, MINOS 5.1 user's guide, Report SOL 83-20R, Department of Operations Research, Stanford University, Stanford, California, 1987.

[40] S. G. Nash and J. Nocedal, A numerical study of the limited memory BFGS method and the truncated-Newton method for large-scale optimization, *SIAM J. Optim.* 1, 358–372, 1991.

[41] S. G. Nash and A. Sofer, A barrier method for large-scale constrained optimization, Report 91-10, Department of Operations Research and Applied Statistics, George Mason University, Fairfax, Virginia, 1991.

[42] J. Nocedal and M. L. Overton, Projected Hessian updating algorithms for nonlinearly constrained optimization, *SIAM J. Numer. Anal.* 22, 821–850, 1985.

[43] C. C. Paige and M. A. Saunders, Solutions of sparse indefinite systems of linear equations, *SIAM J. Numer. Anal.* 12, 617–629, 1975.

[44] C. R. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[45] M. J. D. Powell, Variable metric methods for constrained optimization, in *Mathematical programming: the state of the art* (A. Bachem, M. Grötschel and B. Korte, eds.), Springer-Verlag, Berlin and New York, 288–311, 1983.

[46] F. J. Prieto, Sequential quadratic programming algorithms for optimization, Ph. D. thesis, Department of Operations Research, Stanford University, 1989.

[47] U. T. Ringertz, Optimal design of nonlinear shell structures, Report FFA TN 91-18, Aeronautical Research Institute of Sweden, 1991.

[48] R. B. Schnabel and E. Eskow, A new modified Cholesky factorization, *SIAM J. Sci. Stat. Comp.* 11, 1136–1158, 1990.

[49] D. C. Sorensen, Collinear scaling and sequential estimation in sparse optimization algorithms, *Math. Prog. Study* 18, 135–159, 1982.

[50] K. A. Williamson, A robust trust region algorithm for nonlinear programming, Report TR 90-22, Department of Mathematical Sciences, Rice University, Houston, Texas, 1990 (revised 1991).

[51] M. H. Wright, *Numerical methods for nonlinearly constrained optimization*, Ph. D. thesis, Computer Science Department, Stanford University, 1976.

[52] M. H. Wright, Interior methods for constrained optimization, in *Acta numerica 1992* (A. Iserles, ed.) Cambridge University Press, New York, 341–407, 1992.

[53] M. H. Wright, Determining subspace information from the Hessian of a barrier function, Report NAM 92-02, AT&T Bell Laboratories, Murray Hill, New Jersey, 1992.