Note on Constrained Optimization and Available Packages

Z. A. Reza (zreza@ualberta.ca)
Department of Civil & Environmental Engineering, University of Alberta

C. V. Deutsch (CDeutsch@civil.ualberta.ca)
Department of Civil & Environmental Engineering, University of Alberta

Abstract

This note briefs on theory of constrained optimization techniques and avialable softwares. The list of algorithms and the packages mentioned is not exhaustive. Many of our engineering problems need to solve constrained optimization subproblems. It was thus felt useful to compile some of these techniques to the group's attention. However, it should be noted that none of the remarks mentioned are authoritative in nature. These algorithms mainly search for locally optimal solutions. Global optimization, stochastic optimization or techniques involving evolutionary computing have not been included.

1 Introduction

The basic idea of an optimization problem is to maximize or minimize an objective function. These subproblems arise in almost every aspect of life and engineering starting from resource allocation to dispatching systems to even parameter estimation problems. Some of these problems impose inherent constraints that limit the solution space further. These problems with constraints are what one knows as constrained optimization problems. Here in a slightly esoteric language the theory behind these problems are laid out along with some discussion on available implementations of the theories.

Discussion starts from generalized problems (most difficult) to specialized problems following some assumptions. Not all the possible techniques have been discoursed. Only those considered to be useful with our own problems have been expatiated. Thus, the ensuing sections talk about

1. Nonlinearly Constrained Programming
2. Bound Constrained Programming
3. Nonlinear Least Squares Problems with Constraints

There are numerous literature available on constrained optimization theory [2, 5, 6, 10, 12, 14].

2 Nonlinearly Constrained Programming

The general constrained optimization problem is to minimize a nonlinear function subject to nonlinear constraints. Two equivalent formulations of this problem are useful for describing algorithms. They are

$$\min\{f(x) : c_i(x) \leq 0, i \in \mathcal{I}, c_i(x) = 0, i \in \mathcal{E}\}, \tag{1}$$

where each c_i is a mapping from $\mathbb{R}^n$ to $\mathbb{R}$, and $\mathcal{I}$ and $\mathcal{E}$ are index sets for inequality and equality constraints, respectively; and

$$\min\{f(x) : c(x) = 0, l \leq x \leq u\}, \tag{2}$$

where c maps $\mathbb{R}^n$ to $\mathbb{R}$, and the lower- and upper-bound vectors, l and u, may contain some infinite components.

The main techniques used for solving constrained optimization problems are reduced-gradient methods, sequential linear and quadratic programming methods, and methods based on augmented Lagrangians and exact penalty functions. Fundamental to the understanding of these algorithms is the Lagrangian function, which for formulation 1 is defined as

$$\mathcal{L}(x, \lambda) = f(x) + \sum_{i \in \mathcal{I} \cup \mathcal{E}} \lambda_i c_i(x).$$

The Lagrangian is used to express first-order and second-order conditions for a local minimizer. To simplify matters just first-order necessary and second-order sufficiency conditions are stated here without trying to make the weakest possible assumptions.

The first-order necessary conditions for the existence of a local minimizer x^* of the constrained optimization problem 1 require the existence of Lagrange multipliers λ_i^*, such that

$$\Delta_x \mathcal{L}(x^*, \lambda^*) = \Delta f(x^*) + \sum_{i \in \mathcal{A}^*} \lambda_i^* \Delta c_i(x^*) = 0,$$

where

$$\mathcal{A}^* = \{i \in \mathcal{I} : c_i(x^*) = 0\} \cup \mathcal{E}$$

is the *active set* at x^*, and $\lambda_i^* \geq 0$ if $i \in \mathcal{A}^* \cap \mathcal{I}$. This result requires a *constraint qualification* to ensure that the geometry of the feasible set is adequately captured by a linearization of the constraints about x^*. A standard constraint qualification requires the constraint normals, $\Delta c_i(x^*)$ for $i \in \mathcal{A}^*$, to be linearly independent.

The second-order sufficiency condition requires that (x^*, λ^*) satisfies the first-order condition and that the Hessian of the Lagrangian

$$\Delta_{xx}^2 \mathcal{L}(x^*, \lambda^*) = \Delta^2 f(x^*) + \sum_{i \in \mathcal{A}^*} \lambda_i^* \Delta^2 c_i(x^*)$$

satisfies the condition

$$\omega^T \Delta_{xx}^2 \mathcal{L}(x^*, \lambda^*) \omega > 0$$

for all nonzero ω in the set

$$\left\{ \omega \in \mathbb{R}^n : \Delta c_i(x^*)^T \omega = 0, i \in \mathcal{I}_+^* \cup \mathcal{E}, \Delta c_i(x^*)^T \omega \leq 0, i \in \mathcal{I}_0^* \right\},$$

where

$$\mathcal{I}_+^* = \{i \in \mathcal{A}^* \cap \mathcal{I} : \lambda_I^* > 0\}, \qquad \mathcal{I}_0^* = \{i \in \mathcal{A}^* \cap \mathcal{I} : \lambda_I^* = 0\}.$$

The previous condition guarantees that the optimization problem is well behaved near x^*; in particular, if the second-order sufficiency condition holds, then x^* is a strict local minimizer of the constrained problem 1. An important ingredient in the convergence analysis of a constrained algorithm is its behavior in the vicinity of a point (x^*, λ^*) that satisfies the second-order sufficiency condition.

The following sections discusses briefly

- sequential quadratic programming
- reduced-gradient methods
- methods based on augmented Lagrangians
- feasible sequential quadratic programming

2.1 Sequential Quadratic Programming

The *sequential quadratic programming* (sequential QP) algorithm is a generalization of Newton's method for unconstrained optimization in that it finds a step away from the current point by minimizing a quadratic model of the problem. In its purest form, the sequential QP algorithm replaces the objective function with the quadratic approximation

$$q_k(d) = \Delta f(x_k)^T d + \frac{1}{2} d^T \Delta_{xx}^2 \mathcal{L}(x_k, \lambda_k) d$$

and replaces the constraint functions by linear approximations. For the formulation 1, the step d_k is calculated by solving the quadratic subprogram

$$\min \left\{ q_k(d) : c_i(x_k) + \Delta c_i(x_k)^T d \leq 0, i \in \mathcal{I} \qquad c_i(x_k) + \Delta c_i(x_k)^T d = 0, i \in \mathcal{E} \right\}. \tag{3}$$

The local convergence properties of the sequential QP approach are well understood when (x^*, λ^*) satisfies the second-order sufficiency conditions. If the starting point x_0 is sufficiently close to x^*, and the Lagrange multiplier estimates $\{\lambda_k\}$ remain sufficiently close to λ^*, then the sequence generated by setting $x_{k+1} = x_k + d_k$ converges to x^* at a second-order rate. These assurances cannot be made in other cases. Indeed, codes based on this approach must modify the subproblem 3 when the quadratic q_k is unbounded below on the feasible set or when the feasible region is empty.

The Lagrange multiplier estimates required to set up the second-order term in q_k can be obtained by solving an auxiliary problem or by simply using the optimal multipliers for the quadratic subproblem at the previous iteration. Although the first approach can lead to more accurate estimates, most codes use the second approach.

The strategy based on subproblem 3 makes the decision about which of the inequality constraints appear to be active at the solution internally during the solution of the quadratic program. A somewhat different algorithm is obtained by making this decision prior to formulating the quadratic program. This variant explicitly maintains a working set Ω_k of apparently active indices and solves the quadratic programming problem

$$\min \left\{ q_k(d) : c_i(x_k) + \Delta c_i(x_k)^T d = 0, i \in \Omega_k \right\} \tag{4}$$

to find the step d_k. The contents of Ω_k are updated at each iteration by examining the Lagrange multipliers for the subproblem 4 and by examining the values of $c_i(x_{k+1})$ at the new iterate x_{k+1} for $i \notin \Omega_k$. This approach is usually called the EQP (equality-based QP) variant of sequential QP, to distinguish it from the IQP (inequality-based QP) variant described above.

The sequential QP approach outlined above requires the computation of $\Delta^2_{xx}\mathcal{L}(x_k, \lambda_k)$. Most codes replace this matrix with the BFGS approximation B_k, which is updated at each iteration. An obvious update strategy (consistent with the BFGS update for unconstrained optimization) would be to define

$$s_k = x_{k+1} - x_k, \qquad y_k = \Delta_x \mathcal{L}(x_{k+1}, \lambda_k) - \Delta_x \mathcal{L}(x_k, \lambda_k)$$

and update the matrix B_k by using the BFGS formula

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}.$$

However, one of the properties that make Broyden-class methods appealing for unconstrained problems - its maintenance of positive definiteness in B_k-is no longer assured, since $\Delta^2_{xx}\mathcal{L}(x^*, \lambda^*)$ is usually positive definite only in a subspace. This difficulty may be overcome by modifying y_k. Whenever $y_k^T s_k$ is not sufficiently positive, y_k is reset to

$$y_k \leftarrow \theta_k y_k + (1 - \theta_k) B_k s_k,$$

where $\theta_k \in [0, 1)$ is the number closest to 1 such that $y_k^T s_k \geq \sigma s_K^T B_k s_k$ for some $\sigma \in (0, 1)$.

The convergence properties of the basic sequential QP algorithm can be improved by using a line search. The choice of distance to move along the direction generated by the subproblem is not as clear as in the unconstrained case, where we simply choose a step length that approximately minimizes f along the search direction. For constrained problems we would like the next iterate not only to decrease f but also to come closer to satisfying the constraints. Often these two aims conflict, so it is necessary to weigh their relative importance and define a *merit* or *penalty function*, which we can use as a criterion for determining whether or not one point is better than another. The ℓ_1 merit function

$$\mathcal{P}_1(x, \nu) = f(x) + \sum_{i \in \mathcal{E}} \nu_i \mid c_i(x) \mid + \sum_{i \in \mathcal{I}} \nu_i \max(c_i(x), 0), \tag{5}$$

where $\nu_i > 0$ are penalty parameters, is used in some codes, while the *augmented Lagrangian* merit function

$$\mathcal{L}_A(x, \lambda; \nu) = f(x) + \sum_{i \in \mathcal{E}} \lambda_i c_i(x) + \frac{1}{2} \sum_{i \in \mathcal{E}} \nu_i c_i^2(x) + \frac{1}{2} \sum_{i \in \mathcal{I}} \psi_i(x, \lambda; \nu),$$

where

$$\psi_i(x, \lambda; \nu) = \frac{1}{\nu_i} \left\{ \max\{0, \lambda_i + \nu_i c_i(x)\}^2 - \lambda_i^2 \right\},$$

is used in others. Alternatively, some codes for equality-constrained problems (for which $\mathcal{I} = \emptyset$) use the merit function

$$f(x) + \sum_{i \in \mathcal{E}} \lambda_i c_i(x) + \left(\sum_{i \in \mathcal{E}} \nu_i c_i^2(x) \right)^{1/2},$$

which combines features of $\mathcal{P}_1$ and $\mathcal{L}_A$.

An important property of the ℓ_1 merit function is that if (x^*, λ^*) satisfies the second-order sufficiency condition, then x^* is a local minimizer of $\mathcal{P}_1$, provided the penalty parameters are

chosen so that $\nu_i > |\lambda_i^*|$. Although this is an attractive property, the use of $\mathcal{P}_1$ requires care. The main difficulty is that $\mathcal{P}_1$ is not differentiable at any x with $c_i(x) = 0$. Another difficulty is that although x^* is a local minimizer of $\mathcal{P}_1$, it is still possible for the function to be unbounded below. Thus, minimizing $\mathcal{P}_1$ does not always lead to a solution of the constrained problem.

The merit function $\mathcal{L}_A$ has similar properties. If (x^*, λ^*) satisfies the second-order sufficiency condition and $\lambda = \lambda^*$, then x^* is a local minimizer of $\mathcal{P}_1$, provided the penalty parameters ν_i are sufficiently large. If $\lambda \neq \lambda^*$, then we can say only that $\mathcal{L}_A$ has a minimizer $x(\lambda)$ near x^* and that $x(\lambda)$ approaches x^* as λ converges to λ^*. Note that in contrast to $\mathcal{P}_1$, the merit function $\mathcal{L}_A$ is differentiable. The Hessian matrix of $\mathcal{L}_A$ is discontinuous at any x with $\lambda_i + \nu_i c_i(x) = 0$ for $i \in \mathcal{I}$, but, at least in the case $\mathcal{I}_0^* = \emptyset$, these points tend to occur far from the solution.

Given an iterate x_k and the search direction d_k, some algorithm $x_{k+1} = x_k + \alpha_k d_k$, where the step length α_k approximately minimizes $\mathcal{P}_1(x_k + \alpha d_k; \nu)$. If the merit function $\mathcal{L}_A$ is selected, the step length α_k is chosen to approximately minimize $\mathcal{L}_A(x_k + \alpha d_k, \lambda_k + \alpha(\lambda_{k+1} - \lambda_k); \nu)$ where d_k is a solution of the quadratic programming subproblem 3 and λ_{k+1} is the associated Lagrange multiplier.

Fletcher's book [10] contains discussions of constrained optimization theory and sequential quadratic programming.

2.2 Augmented Lagrangian Methods

Augmented Lagrangian algorithms are based on successive minimization of the augmented Lagrangian $\mathcal{L}_A$ with respect to x, with updates of λ and possibly ν occurring between iterations. An augmented Lagrangian algorithm for the constrained optimization problem computes x_{k+1} as an approximate minimizer of the subproblem

$$\min\{\mathcal{L}_A(x, \lambda_k; \nu_k) : l \leq x \leq u\},$$

where

$$\mathcal{L}_A(x, \lambda; \nu) = f(x) + \sum_{i \in \mathcal{E}} \lambda_i c_i(x) + \frac{1}{2} \sum_{i \in \mathcal{E}} \nu_i c_i^2(x)$$

includes only the equality constraints. Updating of the multipliers usually takes the form

$$\lambda_i \leftarrow \lambda_i + \nu_i c_i(x_k).$$

This approach is relatively easy to implement because the main computational operation at each iteration is minimization of the smooth function $\mathcal{L}_A$ with respect to x, subject only to bound constraints. Bertsekas [2] has an excellent treatment of augmented Lagrangian algorithms.

2.3 Reduced-Gradient Method

Reduced-gradient algorithms avoid the use of penalty parameters by searching along curves that stay near the feasible set. Essentially, these methods take Subproblem 2 and use the equality constraints to eliminate a subset of the variables, thereby reducing the original problem to a bound-constrained problem in the space of the remaining variables. If x_B is the vector of eliminated or *basic* variables, and x_N is the vector of *nonbasic* variables, then

$$x_B = h(x_N),$$

where the mapping h is defined implicitly by the equation

$$c[h(x_N), x_N] = 0.$$

In practice,

$$x_B = h(x_N)$$

can be recalculated using Newton's method whenever x_N changes. Each Newton iteration has the form

$$x_B \leftarrow x_B - \partial_B c(x_B, x_N)^{-1} c(x_B, x_N),$$

where $\partial_B c$ is the Jacobian matrix of c with respect to the basic variables. The original constrained problem is now transformed into the bound-constrained problem

$$\min\{f(h(x_N), x_N) : l_N \leq x_N \leq u_N\}.$$

Algorithms for this reduced subproblem subdivide the nonbasic variables into two categories. These are the *fixed* variables x_F, which usually include most of the variables that are at either

their upper or lower bounds and that are to be held constant on the current iteration, and the *superbasic* variables x_S, which are free to move on this iteration. The standard reduced-gradient algorithm searches along the steepest-descent direction in the superbasic variables. The generalized reduced-gradient codes use more sophisticated approaches. They either maintain a dense BFGS approximation of the Hessian of f with respect to x_S or use limited-memory conjugate gradient techniques. Some codes do not apply the reduced-gradient algorithm directly to problem 1, but rather uses it to solve a linearly constrained subproblem to find the next step. The overall technique is known as a *projected augmented Lagrangian* algorithm.

Operations involving the inverse of $\partial_B c(x_B, x_N)$ are frequently required in reduced-gradient algorithms. These operations are facilitated by an LU factorization of the matrix. Some codes perform a dense factorization, while others use sparse factorization techniques, making them more suitable for large-scale problems.

When some of the components of the constraint functions are linear, most algorithms aim to retain feasibility of all iterates with respect to these constraints. The optimization problem becomes easier in the sense that there is no curvature term corresponding to these constraints that must be accounted for and, because of feasibility, these constraints make no contribution to the merit function. Numerous codes are able to take advantage of linearity in the constraint set. Other codes are specifically designed for linearly constrained problems. Codes based on a sequential quadratic programming algorithm combines features of the EQP and IQP variants. At each iteration, this algorithm determines a set $\mathcal{N}_k$ of near-active indices defined by

$$\mathcal{N}_k = \{i \in \mathcal{I} : c_i(x_k) \geq -\tau_i\},$$

where the tolerances τ_i tend to decrease on later iterations. The step d_k is obtained by solving the subproblem

$$\min \left\{ q_k(d) : c_i(x_k + d_k) = 0, i \in \mathcal{E}, c_i(x_k + d_k) \leq c_i(x_k), i \in \mathcal{N}_k \right\},$$

where

$$q_k(d) = \Delta f(x_k)^T d + \frac{1}{2} d^T B_k d,$$

and B_k is a BFGS approximation to $\Delta^2 f(x_k)$. This algorithm is designed to avoid the short steps that EQP methods sometimes produce, without taking many unnecessary constraints into account, as IQP methods do. The book by Gill, Murray, and Wright [12] discusses reduced-gradient methods.

2.4 Feasible Sequential Quadratic Programming

Finally, we mention *feasible sequential quadratic programming* algorithms, which, as their name suggests, constrain all iterates to be feasible. They are more expensive than standard sequential QP algorithms, but they are useful when the objective function f is difficult or impossible to calculate outside the feasible set, or when termination of the algorithm at an infeasible point (which may happen with most algorithms) is undesirable. The code FSQP solves problems of the form

$$\min\{f(x) : c(x) \leq 0, Ax = b\}.$$

In this algorithm, the step is defined as a combination of the sequential QP direction, a strictly feasible direction (which points into the interior of the feasible set) and, possibly, a second-order correction direction. This mix of directions is adjusted to ensure feasibility while retaining fast local convergence properties. Feasible algorithms have the additional advantage that the objective function f can be used as a merit function, since, by definition, the constraints are always satisfied. FSQP also solves problems in which f is not itself smooth, but is rather the maximum of a finite set of smooth functions

$$f_i : \mathbb{R}^n \to \mathbb{R}.$$

3 Bound Constrained Optimization

Bound-constrained optimization problems play an important role in the development of software for the general constrained problem because many constrained codes reduce the solution of the general problem to the solution of a sequence of bound-constrained problems. The development of software for this problem, which we state as

$$\min\{f(x) : l \leq x \leq u\},$$

is also important in applications because parameters that describe physical quantities are often constrained to lie in a given range.

Algorithms for the solution of bound-constrained problems seek a local minimizer x^* of f. The standard first-order necessary condition for a local minimizer x^* can be expressed in terms of the *binding* set

$$\mathcal{B}(x^*) = \{i : x_i^* = l_i, \partial_i f(x^*) \geq 0\} \cup \{i : x_i^* = u_i, \partial f(x^*) \leq 0\}$$

at x^* by requiring that

$$\partial_i f(x^*) = 0, \qquad i \notin \mathcal{B}(x^*).$$

There are other ways to express this condition, but this form brings out the importance of the binding constraints. A second-order sufficient condition for x^* to be a local minimizer of the bound-constrained problem is that the first-order condition hold and that

$$\omega^T \Delta^2 f(x^*) \omega > 0$$

for all vectors ω with $\omega \neq 0, \quad \omega_i = 0, i \in \mathcal{B}_s(x^*)$, where

$$\mathcal{B}_s(x^*) = \mathcal{B}(x^*) \cap \{i : \partial_i f(x^*) \neq 0\}$$

is the *strictly binding* set at x^*.

Given any set of *free* variables F, we can define the *reduced gradient* and the *reduced Hessian* matrix, respectively, as the gradient of f and the Hessian matrix of f with respect to the free variables. In this terminology, the second-order condition requires that the reduced gradient be zero and that the reduced Hessian matrix be positive definite when the set F of free variables consists of all the variables that are not strictly binding at x^*. As we shall see, algorithms for the solution of bound-constrained problems use unconstrained minimization techniques to explore the reduced problem defined by a set F_k of free variables. Once this exploration is complete, a new set of free variables is chosen with the aim of driving the reduced gradient to zero.

The following sections discusses briefly

- Newton Methods
- Gradient-Projection Methods

Fletcher's book [10] gives a good explanation of Newton and quasi-Newton Methods.

3.1 Newton Methods

Most codes implement line-search and trust-region versions of unconstrained minimization algorithms. Thus, only the differences between the unconstrained and bound-constrained cases is discussed here.

A line-search method for bound-constrained problems generates a sequence of iterates by setting

$$x_{k+1} = x_k + \alpha_k d_k,$$

where x_k is a feasible approximation to the solution, d_k is a search direction, and $\alpha_k > 0$ is the step. The direction d_k is obtained as an approximate minimizer of the subproblem

$$\min \left\{ \Delta f(x_k)^T d + \frac{1}{2} d^T B_k d : d_i = 0, i \in \Omega_k \right\}, \tag{6}$$

where Ω_k is the *working* set and B_k is an approximation to the Hessian matrix of f at x_k. All variables in the working set Ω_k are fixed during this iteration, while all other variables are in the free set F_k. We can express this subproblem in terms of the free variables by noting that it is equivalent to the unconstrained problem

$$\min \left\{ g_k^T \omega + \frac{1}{2} \omega^T A_k \omega : \omega \in \mathbb{R}^{m_k} \right\},$$

where m_k is the number of free variables, A_k is the matrix obtained from B_k by taking those rows and columns whose indices correspond to the free variables, and g_k is obtained from $\Delta f(x_k)$ by taking the components whose indices correspond to the free variables,

The main requirement on Ω_k is that d_k be a feasible direction, that is, $x_k + \alpha d_k$ satisfies the constraints for all $\alpha > 0$ sufficiently small. This is certainly the case if $\Omega_k = \mathcal{A}(x_k)$, where

$$\mathcal{A}(x) = \{i : x_i = l_i\} \cup \{i : x_i = u_i\}$$

is the set of *active* constraints at x. As long as progress is being made with the current Ω_k, the next working set Ω_{k+1} is obtained by merging $\mathcal{A}(x_{k+1})$ with Ω_k. This updating process is continued until the function cannot be reduced much further with the current working set. At this point, the classical strategy is to drop a constraint in Ω_k for which $\partial_i f(x_k)$ has the wrong sign, that is, $i \in \Omega_k$ but $i \notin \mathcal{B}(x_k)$, where the binding set

$$\mathcal{B}(x) = \{i : x_i = l_i, \partial_i f(x) \geq 0\} \cup \{i : x_i = u_i, \partial_i f(x) \leq 0\}$$

is defined as before. In general it is advantageous to drop more than one constraint, in the hope that the algorithm will make more rapid progress towards the optimal binding set. However, all dropping strategies are constrained by the requirement that the solution d_k of the subproblem be a feasible direction.

An implementation of a line-search method based on subproblem 6 must cater to the situation in which the reduced Hessian matrix A_k is indefinite, because in this case the subproblem does not have a solution. This situation may arise, for example, if B_k is the Hessian matrix or an approximation obtained by differences of the gradient. Here, it is necessary to specify d_k by other means. For example, we can use the modified Cholesky factorization.

Quasi-Newton methods for bound-constrained problems update an approximation to the reduced Hessian matrix since, as already noted, only the reduced Hessian matrix is likely to be positive definite. The updating process is not entirely satisfactory because there are situations in which a positive definite update that satisfies the quasi-Newton condition does not exist. Moreover, complications arise because the dimension of the reduced matrix changes when the working set Ω_k changes. Quasi-Newton methods are usually beneficial when the working set remains fixed during consecutive iterations.

The choice of line-search parameter α_k is quite similar to the unconstrained case. If subproblem 6 has a solution d_k and $x_k + d_k$ violates one of the constraints, then we compute the largest $\mu_k \in (0, 1)$ such that

$$x_k \mu_k d_k$$

is feasible. A standard strategy for choosing α_k is to seek an $\alpha_k \in (0, \mu_k]$ that satisfies the sufficient decrease and curvature conditions. We are guaranteed the existence of such an α_k unless μ_k satisfies the sufficient decrease condition and

$$\Delta f(x_k + \mu_k d_k)^T d_k < 0.$$

This situation is likely to happen if, for example, f is strictly decreasing on the line segment $[x_k, x_k + \mu_k d_k]$. In this case it is safe to set $\alpha_k = \mu_k$.

3.2 Gradient-Projection Methods

Active set methods have been criticized because the working set changes slowly; at each iteration at most one constraint is added to or dropped from the working set. If there are k_0 constraints active at the initial Ω_0, but k_θ constraints active at the solution, then at least $\mid k_\theta - k_0 \mid$ iterations are required for convergence. This property can be a serious disadvantage in large problems if the working set at the starting point is vastly different from the active set at the solution. Consequently, recent investigations have led to algorithms that allow the working set to undergo radical changes at each iteration and to interior-point algorithms that do not explicitly maintain a working set.

The *gradient-projection* algorithm is the prototypical method that allows large changes in the working set at each iteration. Given x_k, this algorithm searches along the piecewise linear path

$$P[x_k - \alpha \Delta f(x_k)], \qquad \alpha \geq 0,$$

where P is the projection onto the feasible set. A new point

$$x_{k+1} = P[x_k - \alpha_k \Delta f(x_k)]$$

is obtained when a suitable $\alpha_k > 0$ is found. For bound-constrained problems, the projection can be easily computed by setting

$$[P(x)]_i = \mid \{x_i, l_i, u_i\} \mid,$$

where $\mid \{\cdot\} \mid$ is the middle (median) element of a set. The search for α_k has to be done carefully since the function

$$\phi(\alpha) = f\left(P[x_k - \alpha \Delta f(x_k)]\right)$$

is only piecewise differentiable.

If properly implemented, the gradient-projection method is guaranteed to identify the active set at a solution in a finite number of iterations. After it has identified the correct active set, the gradient-projection algorithm reduces to the steepest-descent algorithm on the subspace of free variables. As a result, this method is invariably used in conjunction with other methods with faster rates of convergence.

Trust-region algorithms can be extended to bound-constrained problems. The main difference between the unconstrained and the bound-constrained version is that we now require the step s_k to be an approximate solution of the subproblem

$$\min\{q_k(s) :\| D_k s \| \leq \Delta_k, l \leq x_k + s \leq u\},$$

where

$$q_k(s) = \Delta f(x_k)^T s + \frac{1}{2} s^T B_k s.$$

An accurate solution to this subproblem is not necessary, at least on early iterations. Instead, we use the gradient-projection algorithm to predict a step s_k^C (the *Cauchy step*) and then require merely that our step, s_k, satisfies the constraints in the trust-region subproblem with $q_k(s_k) \leq q_k(s_k^C)$. In some bound-constrained code, the trust region is defined by the L_∞-norm and $D_k = T$, yielding the equivalent subproblem

$$\min\{q_k(s) : \max(l - x_k, \Delta_k e) \leq s \leq \min(u - x_k, \Delta_k e)\},$$

where e is the vector of all ones.

The advantage of strategies that combine the gradient-projection method with trust-region methods is that the working set is allowed to change rapidly, and yet eventually settle into the working set for the solution. Using this approach, together with special data structures that exploit the partially separable structure of f, large bound-constrained problems can be solved.

4 Nonlinear Least Squares Problems with Constraints

The nonlinear least squares problem with constraints has the form

$$\min\{r(x) : x \in \mathbb{R}^n\},$$

$$\text{subject to } l \leq x \leq u$$

where r is the function defined by $r(x) = \frac{1}{2} \| f(x) \|_2^2$ for some vector-valued function f that maps $\mathbb{R}^n$ to $\mathbb{R}^m$, and l and u are the lower and upper bounds on the vector x, respectively.

Consider some physical process modeled by a nonlinear function that depends on a parameter vector x and time t. If b_i is the actual output of the system at time t_i, then the residual

$$\phi(x, t_i) - b_i$$

measures the discrepancy between the predicted and observed outputs of the system at time t_i. A reasonable estimate for the parameter x may be obtained by defining the ith component of f by

$$f_i(x) = \phi(x, t_i) - b_i,$$

and solving the least squares problem with this definition of f.

From an algorithmic point of view, the feature that distinguishes least squares problems (without constraints) from the general unconstrained optimization problem is the structure of the Hessian matrix of r. The Jacobian matrix of f, $f'(x) = (\partial_1 f(x), \ldots, \partial_n f(x))$, can be used to express the gradient of r since $\Delta r(x) = f'(x)^T f(x)$. Similarly, $f'(x)$ is part of the Hessian matrix $\Delta^2 r(x)$ since $\Delta^2 r(x) = f'(x)^T f'(x) + \sum_{i=1}^m f_i(x) \Delta^2 f_i(x)$. To calculate the gradient of r, we need to calculate the Jacobian matrix $f'(x)$. Having done so, we know the first term in the Hessian matrix $\Delta^2 r(x)$ without doing any further evaluations. Nonlinear least squares algorithms exploit this structure.

In many practical circumstances, the first term $f'(x)^T f'(x)$ in $\Delta^2 r(x)$ is more important than the second term, most notably when the residuals $f_i(x)$ are small at the solution. Specifically, we say that a problem has small residuals if, for all x near a solution, the quantities

$$| f_i(x) | \| \Delta^2 f_i(x) \|, \qquad i = 1, \ldots, n$$

are small relative to the smallest eigenvalue of $f'(x)^T f'(x)$.

Constrained least squares problems can be solved with the methods for bound constrained problems, possibly modified to take advantage of the special Hessian approximations that are available for nonlinear least squares problems. Active set methods for handling the bounds form part of the capability of some of these codes. An approach based on the gradient-projection method, which is more suitable for large-scale applications, is used by others. Yet other algorithms use active set versions of the Levenberg-Marquardt algorithm, as well as of the hybrid strategy that combines the Gauss-Newton and BFGS quasi-Newton algorithms.

Some implementations use the same sequential quadratic programming strategy as the general nonlinear programming, but also make use of the Jacobian matrix $f'(x)$ to compute a starting approximation to the Hessian of the Lagrangian for the constrained problem and to calculate the gradient Δr. Use of sequential quadratic programming techniques while exploiting the structure of r in its choice of approximate Hessian is also possible.

5 Some Important Constrained Optimization Packages

5.1 LBFGS-B

L-BFGS-B is a limited memory algorithm for solving large nonlinear optimization problems subject to simple bounds on the variables. It is intended for problems in which information on the Hessian matrix is difficult to obtain, or for large dense problems. L-BFGS-B can also be used for unconstrained problems [28].

Authors: Ciyou Zhu, Richard H. Byrd, Peihuang Lu and Jorge Nocedal; Language: FORTRAN

5.2 TRON

TRON is a trust region Newton method for the solution of large bound-constrained optimization problems. TRON uses a gradient projection method to generate a Cauchy step, a preconditioned conjugate gradient method with an incomplete Cholesky factorization to generate a direction, and a projected search to compute the step. The use of projected searches, in particular, allows TRON to examine faces of the feasible set by generating a small number of minor iterates, even for problems with a large number of variables. As a result TRON is remarkably efficient at solving large bound-constrained optimization problems [15].

TRON has the advantages of (1) no assumptions of strict complementarity, (2) global convergence; fast local convergence, (3) identification of optimal face in a finite number of iterations, and (4) an incomplete Cholesky factorization with predictable storage requirements.

Authors: C. Lin and J. More; Language: FORTRAN

5.3 TOLMIN

TOLMIN is dedicated for minimization of a differentiable function of several variables, subject to linear constraints (equality and/or inequality) and simple bounds on variables. It is based on successive quadratic programming algorithm. Each search direction is calculated so that it does not intersect the boundary of any inequality constraint that is satisfied and that has a "small" residual at the beginning of the line search. The meaning of "small" depends on a parameter called TOL which is automatically adjusted, and which gives the name of the software [19].

Author: M.J.D. Powell; Language: FORTRAN

5.4 OPTPACK

The package [13] has three main subroutines: (1) **min** - to minimize a function subject to nonlinear constraints, (2) **bmin** - to minimize a function subject to simple bounds on variables, and (3) **lmin** - to minimize a function subject to linear constraints. Unconstrained optimization is performed using the conjugate gradient algorithm. Constrained optimization is performed using a new scheme that combines multiplier methods with preconditioning and linearization techniques to accelerate convergence.

Author: William W. Hager (hager@math.ufl.edu); Language: FORTRAN

5.5 NITRO

A barrier approach is used that employs sequential quadratic programming and trust regions (interior point trust region) to solve the subproblems occurring in the iteration. Both primal and primal-dual versions of the algorithm are developed [4].

Authors: Richard H. Byrd (richard@cs.colorado.edu), Mary E. Hribar, and Jorge Nocedal (nocedal@ece.nwu.edu); Language: FORTRAN

5.6 NLPQL

NLPQL [20] has the features of having: (1) upper and lower bounds on the variables handled separately, (2) reverse communication (evaluation of function values in main program), (3) scaling of function values, (4) initial multiplier and Hessian approximation, (5) bounds and linear constraints remain satisfied. The internal algorithm is a sequential quadratic programming (SQP) method. Proceeding from a quadratic approximation of the Lagrangian function and a linearization of the constraints, a quadratic subproblem is formulated and solved by the dual code QL. Subsequently a line search is performed with respect to two alternative merit functions and the Hessian approximation is updated by the modified BFGS-formula.

Author: K. Schittkowski (klaus.schittkowski@uni-bayreuth.de); Language: FORTRAN

5.7 NLPQLB

NLPQLB is an extension of the nonlinear programming code NLPQL with the intention to solve also problems with very many constraints, where the derivative matrix of the constraints does not possess any special sparsity structure that can be exploited numerically [21].

Author: K. Schittkowski (klaus.schittkowski@uni-bayreuth.de); Language: FORTRAN

5.8 NLPSPR

The package is dedicated to solve nonlinear programming problems with a large number of variables and constraints where the Jacobian and Hessian matrices are sparse. Sparse linear systems are solved efficiently using a multifrontal algorithm that implements a modified Cholesky decomposition for symmetric indefinite systems. Employs sequential QP algorithm that uses an augmented Lagrangian merit function and a sparse quadratic programming algorithm. The user must supply the sparse Jacobian and Hessian matrices, although this information can be computed efficiently using sparse finite differences which are implemented in a utility package that is also available. The software incorporates a reverse communication structure and is especially well suited for applications derived from discretized optimal control problems [3].

Authors: John T. Betts and Paul D. Frank; Language: FORTRAN

5.9 MINOS

MINOS (Modular Incore Nonlinear Optimization System) is a software package for solving large-scale optimization problems (linear and nonlinear programs). MINOS solves a sequence of subproblems in which the constraints are linearized and the objective is an augmented Lagrangian. It is especially effective for problems with a smooth nonlinear objective function and sparse linear constraints (e.g., quadratic programs). MINOS can also process large numbers of smooth nonlinear constraints. The functions need not be convex. Numerically stable algorithms. Needs only first derivatives. Has warm start capability. MINOS is highly effective for problems with a nonlinear objective function and large numbers of sparse linear constraints (as well as bounds on the variables) [16].

Authors: Bruce A. Murtagh and Michael Saunders (Mike@SOL-Michael.Stanford.edu); Language: FORTRAN

5.10 LSSOL

LSSOL is a software package for solving constrained linear least-squares problems and convex quadratic programs (definite or semidefinite), including linear programs. Dense matrices are assumed throughout. LSSOL is recommended for QP problems whose objective includes a term of the form $x^T A^T A x$ for some matrix A (which may be rectangular, square or triangular).

Linear constraints and bounds on the variables are treated separately by an active-set method. If the problem has no feasible solution, LSSOL minimizes the sum of the constraint and bound violations. LSSOL is used as a subroutine inside NPSOL to solve a sequence of related quadratic programs, using warm starts. Numerically stable algorithms. Warm start capability. Elastic bounds on variables and constraints (for infeasible problems). General-purpose dense linear programming and quadratic programming [11].

Authors: Philip Gill (pgill@ucsd.edu), Walter Murray (Walter@SOL-Walter.Stanford.edu), Michael Saunders (Mike@SOL-Michael.Stanford.edu) and Margaret H. Wright; Language: FORTRAN

5.11 LANCELOT

The LANCELOT package uses an augmented Lagrangian approach to handle all constraints other than simple bounds. The bounds are dealt with explicitly at the level of an outer-iteration subproblem, where a bound-constrained nonlinear optimization problem is approximately solved at each iteration. The algorithm for solving the bounded problem combines a trust region approach adapted to handle the bound constraints, projected gradient techniques, and special data structures to exploit the (group partially separable) structure of the underlying problem [7].

Authors: Andy Conn (arconn@watson.ibm.com), Nick Gould (nimg@ib.rl.ac.uk), and Philippe Toint (pht@math.fundp.ac.be); Language: FORTRAN

5.12 CONOPT

The algorithm in CONOPT is based on the generalized reduced gradient (GRG) algorithm. All matrix operations are implemented by using sparse matrix techniques to allow very large models. Without compromising the reliability of the GRG approach, the overhead of the GRG algorithm is minimized by, for example, using dynamic feasibility tolerances, reusing Jacobians whenever possible, and using an efficient reinversion routine. The algorithm uses many dynamically set tolerances and therefore runs, in most cases, with default parameters.

The system is continuously being updated, mainly to improve reliability and efficiency on large models. The latest additions are options for SLP and steepest edge [8, 9].

Author: Arne Stolbjerg Drud (adrud@arki.dk); Language: FORTRAN

5.13 COPL_LC

The approach is a primal-dual homogeneous algorithm (interior-point homogeneous algorithm). The algorithm generates a solution pair if the problem is solvable, or detects infeasibility or unboundedness. The objective need not be differentiable at an optimal solution of a given consistent program. The package implements advanced sparse matrix factorization techniques to take advantage of the structure of the Hessian of the objective function [1].

Author: Yinyu Ye (yinyu-ye@uiowa.edu); Language: FORTRAN

5.14 DONLP2

DONLP2 is a FORTRAN package designed to solve the nonlinear programming problem: the minimization of a smooth nonlinear function subject to a set of constraints on the variables. Employs sequential quadratic programming method incorporating the exact l_1- merit function and a special BFGS quasi-Newton approximation to the Hessian. Lower and Upper bounds on the variables are identified by a special indicator array. DONLP2 treats all matrices as dense [23, 24].

Author: Peter Spellucci (spellucci@mathematik.tu-darmstadt.de); Language: FORTRAN

5.15 FSQP

Feasible Sequential Quadratic Programming (FSQP) algorithm is a superlinearly convergent algorithm for directly tackling optimization problems with: (1) multiple competing linear/nonlinear objective functions (minimax), (2) linear/nonlinear inequality constraints, and (3) linear/nonlinear equality constraints. The basic problem solved is (where the variable x is n-dimensional) In phase 1 - iterate is generated satisfying all linear constraints and nonlinear inequality constraints. In Phase 2 - maximum of objectives is minimized. Iterates satisfy all constraints except nonlinear

equality constraints (which are asymptotically satisfied). The algorithm contains special provisions for maintaining "semi-feasibility" of each iterate. Efficiently handling problems with many "sequentially related" objectives and/or constraints [27].

Authors: Dr. Eliane R. Panier, Prof. Andre Tits, Jian Zhou, and Craig Lawrence; Language: FORTRAN, C

5.16 OPTIMA Library

Library contains these modules among others [18]: OPVMB - Optimization subject to simple bounds, OPRQP - Sequential quadratic programming, but superseded by the OPXRQP routine, OPXRQP - A more efficient implementation of sequential quadratic programming; uses the EQP variant, OPSQP - Another implementation of sequential quadratic programming, but uses the IQP variant, which gives rise to inequality-constrained subproblems, OPALQP - Similar to OPSQP, but uses an augmented Lagrangian line search function, OPIPF - Sequential minimization of a sequence of augmented Lagrangians.

Author: M. C. Bartholomew-Biggs; Language: FORTRAN

5.17 LOQO

A primal-dual interior-point method is used. For convex problems a globally optimal solution is obtained. For nonconvex ones a locally optimal solution near a given initial solution is found. LOQO is designed to handle thousands of constraints and variables [26].

Author: Robert J. Vanderbei; Language: C, C++, MatLab

5.18 VE08

VE08 is a line search method with a search direction obtained by a truncated conjugate gradient technique. The bounds are handled by bending the search direction on the boundary of the feasible domain. VE08 also contains provision for estimating gradients by finite difference, if they are unavailable, or to check the analytic gradients otherwise. It features both Newton and quasi-Newton algorithms. VE08 exploits the partially separable structure of many large-scale problems to obtain good efficiency. In particular, it uses the partitioned updating technique when a quasi-Newton method is chosen [25].

Author: Ph Toint (pht@math.fundp.ac.be); Language: FORTRAN

5.19 TN TNBC

TN uses a truncated-Newton method based on a line search. Truncated-Newton methods compute an approximation to the Newton direction by approximately solving the Newton equations using an iterative method. In this software, the conjugate gradient method is used as the iterative solver [17].

Author: Stephen G. Nash (snash@gmu.edu); Language: FORTRAN

5.20 LSGRG2

LSGRG2 uses an implementation of the generalized reduced gradient (GRG) algorithm similar to that used in GRG2. However, it uses a sparse data structure to store and manipulate the constraint Jacobian matrix, and a sparse inversion procedure to factor the basis. It can therefore solve large, sparse nonlinear programs: problems with over 500 constraints have been solved successfully [22].

Author: S. Smith and Leon Lasdon; Language: FORTRAN

5.21 SNOPT

SNOPT is suitable for general large-scale QP problems. It is more efficient if only some of the variables appear quadratically in the objective, or if many constraints are active at a solution. If the quadratic is indefinite (not positive definite or semidefinite), the solution obtained may be a local optimum. The SQP algorithm used by SNOPT is highly effective for problems with a nonlinear objective function and large numbers of sparse linear constraints. Efficiency is best if only some of the variables enter nonlinearly, or if the number of active constraints (including

simple bounds) is nearly as large as the number of variables. SNOPT performs best if both function and gradient values can be provided. It uses an SQP algorithm (Sequential Quadratic Programming) in which a sequence of subproblems in which the constraints are linearized, and the objective is a quadratic approximation of a Lagrangian. (Hence, no function or gradient values are needed during the solution of each QP.)

Authors: Philip Gill (pgill@ucsd.edu), Walter Murray (Walter@SOL-Walter.Stanford.edu), Michael Saunders (Mike@SOL-Michael.Stanford.edu); Language: FORTRAN

References

[1] E. D. Andersen and Y. Ye. On a homogeneous algorithm for a monotone complementarity problem with nonlinear equality constraints. In M. C. Ferris and J.-S. Pang, editors, *Complementarity and variational Problems: State of the art*, pages 1–11. SIAM, 1997.

[2] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods.* Academic Press, New York, 1982.

[3] J. T. Betts and P. D. Frank. A sparse nonlinear optimization algorithm. Technical Report AMS-TR-173, Applied Mathematics and Statistics Group, Boeing Computer Services, 1991.

[4] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large scale nonlinear programming. Technical Report OTC 97/05, Optimization Technology Center, Argonne National Laboratory, 1997.

[5] T. F. Coleman. Large scale numerical optimization: Introduction and overview. In J. Williams and A. Kent, editors, *Encyclopedia of Computer Science and Technology*, pages 167–195. Marcel Dekker, New York, 1993.

[6] A. R. Conn, N. I. M. Gould, , and P. L. Toint. Large-scale nonlinear constrained optimization. In R. E. O'Malley, editor, *Proceedings of the Second International Conference on Industrial and Applied Mathematics*, pages 51–70. SIAM, 1992.

[7] A. R. Conn, N. I. M. Gould, and P. L. Toint. *LANCELOT: a FORTRAN Package for Large-Scale Nonlinear Optimization (Release A)*, volume 17 of *Springer Series in Computational Mathematics*. Springer Verlag, New York, 1992.

[8] A. Drud. CONOPT: A GRG code for large sparse dynamic nonlinear optimization problems. *Mathematical Programming*, 31:153–191, 1985.

[9] A. Drud. CONOPT – a large scale grg code. *ORSA Journal on Computing*, 6:207–216, 1994.

[10] R. Fletcher. *Practical Methods of Optimization, 2nd ed.* John Wiley & Sons, Inc, New York, 1987.

[11] P. E. Gill, S. J. Hammarling, W. Murray, M. A. Saunders, and M. H. Wright. User's guide for LSSOL (version 1.0): A fortran package for constrained linear least-squares and convex quadratic programming. Technical Report SOL 86-1, Systems Optimization Laboratory, Stanford University, 1986.

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

[13] W. W. Hager. Analysis and implementation of a dual algorithm for constrained optimization. Technical report, Department of Mathematics, University of Florida, Gainesville, Florida, 1990.

[14] W. W. Hager, R. Horst, , and P. M. Pardalos. Mathematical programming-a computational perspective. In C. R. Rao, editor, *Handbook of Statistics*, pages 201–278. Elsevier, New York, 1993.

[15] C.-J. Lin and J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM J. of Optimization*, 9(4):1100–1127, 1999.

[16] B. A. Murtagh and M. A. Saunders. Large-scale linearly constrained optimization. *Mathematical Programming*, 14:41–72, 1978.

[17] S. G. Nash. User's guide for TN/TNBC. Technical Report 397, Department of Mathematical Sciences, The Johns Hopkins University, Baltimore, 1984.

[18] Numerical Optimisation Center, University of Hertfordshire, UK. *OPTIMA Manual*, 1989.

[19] M. J. D. Powell. TOLMIN: A fortran package for linearly constrained optimization calculation. Technical Report 1989/NA2, DAMTP, 1989.

[20] K. Schittkowski. NLPQL: A fortran subroutine for solving constrained nonlinear programming problems. *Annals of Operations Research*, 5:485–500, 1985.

[21] K. Schittkowski. Solving nonlinear programming problems with very many constraints. *Optimization*, 25:179–196, 1992.

[22] S. Smith and L. Lasdon. Solving large sparse nonlinear programs using grg. *ORSA J. Comput.*, 4:1–15, 1992.

[23] P. Spellucci. Sequential quadratic programming: Theory, implementation, problems. In M. J. Beckmann, K. W. Gaede, K. Ritter, and Schneeweiss, editors, *Methods of Operations Research*, volume 53, pages 183–213. 1985.

[24] P. Spellucci. An sqp method for general nonlinear programs using only equality constrained subproblems. *Mathematical Programming*, 82(3):413–448, 1998.

[25] P. L. Toint. User's guide to the routine ve08 for solving partially separable bounded optimization problems. Technical Report 83/1, FUNDP, Namur, Belgium, 1983.

[26] R. J. Vanderbei. LOQO: An interior point code for quadratic programming. Technical Report SOR 94-15, Princeton University, 1994.

[27] J. L. Zhou, A. L. Tits, and C. T. Lawrence. User's guide for FFSQP version 3.7 : A fortran code for solving optimization programs, possibly minimax, with general inequality constraints and linear equality constraints, generating feasible iterates. Technical Report SRC-TR-92-107r5, Institute for Systems Research, University of Maryland, 1997.

[28] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization. *ACM Trans. Math. Software*, 23(4):550–560, 1997.

Notations

$\mathcal{A}^*$:	active sets at optimal solution x^*
B_k:	BFGS approximation matrix of $\mathcal{L}^2_{xx}(x_k, \lambda_k)$
$c(x)$:	constraints on x that maps $\mathbb{R}^n \to \mathbb{R}$
Δ:	derivative operative
Δ_x:	first partial derivative w.r.t. x
Δ^2_{xx}:	second partial derivative w.r.t. x
$\mathcal{E}$:	index sets of equality constraints
$f(x)$:	function space of x
$\mathcal{I}$:	index sets of inequality constraints
λ:	Lagrangian multipliers
$\mathcal{L}(x, \lambda)$:	Lagrangian function w.r.t. x with Lagrangian multipliers λ
$\mathcal{L}_A$:	augmented Lagrangian function
max:	maximum
min:	minimum
ν_i:	penalty parameters
$\mathbb{R}$:	space of real numbers
$\mathbb{R}^n$:	n-tuple space of real numbers
P:	projection operator in Projection-Gradient methods
$\mathcal{P}(x, \nu)$:	penalty functions of x with penalty parameters ν
$\mathbf{A^T}$:	transpose of any matrix $\mathbf{A}$
$\mathbf{A^{-1}}$:	inverse of any matrix $\mathbf{A}$
$\cup$:	union of two sets
$\cap$:	intersection of two sets