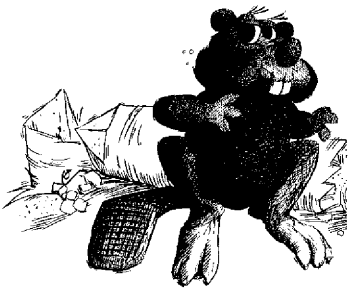
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT



On Generating Test Problems for Nonlinear Programming Algorithms

Richard H. Bartels
Nezam Mahdavi-Amiri

CS-83-12

June, 1983

UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT

On Generating Test Problems for Nonlinear Programming Algorithms

Richard H. Bartels

University of Waterloo
Department of Computer Science
Waterloo, Ontario
Canada N2L 3G1

Nezam Mahdavi-Amiri

York University
Department of Computer Science
Downsview, Ontario
Canada M3J 1P3

ABSTRACT

We present an approach to test-problem generation which provides a way of constructing example nonlinear programming problems from a wide variety of given functions. The approach permits one to specify an arbitrary number of points at each one of which the problem should satisfy some "interesting" conditions (i.e. be optimal or stationary or degenerate) and to determine the characteristics of functions and derivatives at these points (i.e. choose predetermined values, gradients, Hessians and Lagrange multipliers).

We give a sample of results obtained by using our approach to generate test problems for two algorithms to minimize nonlinear-least-squares objectives subject to nonlinear constraints.

This research was supported under NSERC grant number A4076 and under a Laidlaw Fellowship administered by the University of Waterloo Computer Science Department. Part of this work was included with that originally submitted to the Department of Mathematical Sciences of The Johns Hopkins University by the second author as a PhD thesis.

1. Introduction

During the last decade a number of methods have been developed for minimizing general objective functions subject to general nonlinear constraints. These methods are characterized by their properties of global convergence, asymptotic superlinear rates of convergence, and formulation in terms of processes which can be efficiently and accurately implemented as numerical algorithms. Examples would include [4, 5, 14, 15, 24, 25, 28].

No case has to be made for the fact that, in translating these or other methods into mathematical software, a ready supply of test examples can be extremely useful. It is less often recognized that such a supply of test examples can also be used to probe for oversights in the theory, and can therefore be useful in earlier and more fundamental stages of algorithm development. It was with the intention of exploring ways of making the generation of test examples more automatic and more flexible that the present work was done.

In section 2 of this paper we review the commonly-given necessary and sufficient conditions for a point to be an optimizer of a nonlinear programming problem. In section 3 we convert this review into a scheme for constructing nonlinear programming example problems with one specified point as an optimizer and with additional specifications on objective and constraint functions and on dual variables. In section 4 we extend this approach to a method for constructing nonlinear programming problems with any number of critical points: maxima, minima, saddle points, degenerate points, etc. This is done with a view for producing local "scenario" conditions which test the assumptions upon which algorithms are based and which exercise critical sections of computer codes. In section 5 we give a brief survey of two constrained nonlinear least squares programs which were written to explore the usefulness of this test generation approach. In section 6 we give some samples of our experience in constructing problems for these codes.

The idea of building automatic test problem generators is not new. References [2, 17, 19, 20, 22, 26, 29, 30, 34] provide a sampling of the literature. Automatic generation schemes provide a supplement to reference collections of optimization problems; e.g. [3, 8, 16, 31]. Finally, the following references provide some discussions on software testing, and the reporting of test results, from a broader perspective: [6, 7, 21].

The method developed in this paper represents an advance over the test generation methods cited above in that it can specify the characteristics of an arbitrary number of points, in a generally constrained setting, with a wide latitude of sample functions.

2. Optimality Conditions

The minimization problems to be considered are of the form

$$\underset{x}{\text{minimize}}\ \phi_0(x) \tag{2.1}$$

such that

$$\phi_j(x) = 0\ ,\quad j \in \mathbf{J}_E$$

$$\phi_j(x) \geq 0\ ,\quad j \in \mathbf{J}_I\ ,$$

where $\mathbf{J}_E = \{1, \ldots, l\}$ is the index set of equality constraints and $\mathbf{J}_I = \{l+1, \ldots, l+m\}$ is the index set of inequality constraints. We will denote the combined index set by

$$\mathbf{J} = \mathbf{J}_E \cup \mathbf{J}_I\ .$$

Another important index set is that of the *active constraints* at any point x:

$$\mathbf{J}_A(x) = \{\, j \in \mathbf{J}\ ,\ \phi_j(x) = 0 \,\}$$

We first state necessary conditions for a given point to be a solution of problem (2.1). In order that these conditions be applicable, constraint qualifications must hold at the point. Appropriate ones, and the underlying theory, can be found in [11]. In practice, the constraint qualifications given in this reference (or any other of suitable generality) are hard to verify. In order to implement algorithms, it is often assumed that the following more strict constraint qualifications hold:

Nondegeneracy Assumption: Letting $\bar{x}$ stand for the optimizer, as well as for any point encountered by the algorithm,

$$\nabla\phi_j(\bar{x}),\quad j \in J_E \cup \mathbf{J}_A(\bar{x}) \tag{2.2}$$

are linearly independent .

This is more than enough to imply satisfaction of all of the constraint qualifications given in [11].

First Order Necessary Conditions: Let x^* be a solution of (2.1), and let an appropriate constraint qualification be satisfied at x^*. Then there exist

λ_j^*, $j \in \mathbf{J}$ such that

$$\nabla\phi_0(x^*) - \sum_{j\in\mathbf{J}_E} \lambda_j^*\nabla\phi_j(x^*) - \sum_{j\in\mathbf{J}_A(x^*)} \lambda_j^*\nabla\phi_j(x^*) = 0 \qquad (2.3)$$

$$\begin{aligned}
\lambda_j^* &\geq 0 \quad j \in \mathbf{J}_A(x^*) \\
\phi_j(x^*) &= 0 \quad j \in \mathbf{J}_E \\
\phi_j(x^*) &\geq 0 \quad j \in \mathbf{J}_I \\
\lambda_j^*\phi_j(x^*) &= 0 \quad j \in \mathbf{J}_I
\end{aligned}$$

Second Order Necessary Conditions: Let x^* be a solution to problem (2.1) and satisfy an appropriate constraint qualification. Suppose that there exist λ_j^* satisfying the first order necessary conditions. Then

$$\text{for all } z \in \mathbf{R}^n \text{ s.t. } \begin{aligned} z^T\nabla\phi_j(x^*) &= 0 \quad j \in \mathbf{J}_E \\ z^T\nabla\phi_j(x^*) &= 0 \quad j \in \mathbf{J}_A(x^*) \end{aligned}$$

we have

$$z^T[\,\nabla^2\phi_0(x^*) - \sum_{j\in\mathbf{J}_E} \lambda_j^*\nabla^2\phi_j(x^*) - \sum_{j\in\mathbf{J}_A(x^*)} \lambda_j^*\nabla^2\phi_j(x^*)\,]z \geq 0. \qquad (2.4)$$

For x^* to be a strict local minimum of problem (2.1) we need the conditions stated in the following result.

Second Order Sufficient Conditions: Let (x^*,λ^*) satisfy the first order necessary conditions and an appropriate constraint qualification. If for every nonzero $z \in \mathbf{R}^n$ such that

$$z^T\nabla\phi_j(x^*) = 0 \quad j \in J_E$$

$$z^T\nabla\phi_j(x^*) = 0 \quad j \in \mathbf{J}_A(x^*) \text{ and } \lambda_j^* > 0$$

$$z^T\nabla\phi_j(x^*) \geq 0 \quad j \in \mathbf{J}_A(x^*) \text{ and } \lambda_j^* = 0$$

it follows that

$$z^T[\nabla^2\phi_0(x^*) - \sum_{j \in J_E} \lambda_j^*\nabla^2\phi_j(x^*) - \sum_{j \in J_A(x^*)} \lambda_j^*\nabla^2\phi_j(x^*)]z > 0, \qquad (2.5)$$

then x^* is a strict local minimum of problem (2.1).

Definition: The function

$$L(x,\lambda) = \phi_0(x) - \sum_{j\in J} \lambda_j\phi_j(x)$$

which clearly plays a role in the above conditions is known as the *Lagrangian* of the problem. If we regard ∇ and ∇^2 to be differentiation operators with respect to x alone, then

(2.3) can be written briefly as $\nabla L(x^*,\lambda^*) = 0$;
(2.4) says that, on a certain subspace $\nabla^2 L(x^*,\lambda^*)$ is nonnegative definite, and
(2.5) says that, on a certain (slightly smaller) subspace
$\nabla^2 L(x^*,\lambda^*)$ is positive definite .

The first order conditions clearly imply that $\lambda_j = 0$ for all $j \in \mathbf{J}_j \backslash \mathbf{J}_A(x^*)$, hence at (x^*,λ^*) the function L can be written in the more explicit form indicated by the first and second order conditions.

The idea to be presented for generating test problems is straightforward: The functions $\phi_j(x)$, $j \in \mathbf{J} \cup \{0\}$, will be constructed from given functions $q_j(x)$ to which perturbing functions are added. The perturbing functions are chosen so as to force the first and/or second order conditions to hold at a number of selected points. It will be seen that this choice is achieved by the introduction of simple interpolating conditions on the perturbing functions. As for the functions $q_j(x)$, they may be chosen to provide a desired nonlinear characteristic (e.g. exponential growth or some degree of discontinuity).

3. Specifying a Single Point

Consider the problem

$$\min_x \phi_0(x) = q_0(x) + \Omega[\tfrac{1}{2}(x - x^*)^T D(x - x^*)] + h_0^T(x - x^*) + \alpha_0$$

such that

$$\phi_j(x) = q_j(x) + h_j^T(x - x^*) + \alpha_j = 0, \quad j \in \mathbf{J}_E$$

and

$$\phi_j(x) = q_j(x) + h_j^T(x - x^*) + \alpha_j \geq 0, \quad j \in \mathbf{J}_I ,$$

where the $q_j(x)$ are chosen functions from $\mathbf{R}^n$ to $\mathbf{R}^1$, where x^* is the proposed optimizer, where D is an $n \times n$ symmetric matrix, the h_j are vectors in $\mathbf{R}^n$ and the α_j are scalars all to be determined, and where Ω is any function satisfying

$$\Omega'(0) \neq 0 \tag{3.1}$$

The gradient and the Hessian of ϕ_0, respectively, are

$$\nabla\phi_0(x) = \nabla q_0(x) + \Omega'[\tfrac{1}{2}(x - x^*)^T D(x - x^*)]D(x - x^*) + h_0$$

and

$$\nabla^2\phi_0(x) = \nabla^2 q_0(x) + \Omega'[\tfrac{1}{2}(x - x^*)^T D(x - x^*)]D + \Omega''[\tfrac{1}{2}(x - x^*)^T D(x - x^*)]D(x - x^*)(x - x^*)^T D .$$

The gradient and the Hessian of the Lagrangian, respectively, are

$$\nabla L(x,\lambda) = \nabla\phi_0(x) - \sum_{j \in \mathbf{J}} \lambda_j \nabla\phi_j(x) = \nabla\phi_0(x) - \sum_{\lambda_j \neq 0} \lambda_j(\nabla q_j(x) + h_j)$$

and

$$\nabla^2 L(x,\lambda) = \nabla^2\phi_0(x) - \sum_{j \in \mathbf{J}} \lambda_j \nabla^2\phi_j(x) = \nabla^2\phi_0(x) - \sum_{\lambda_j \neq 0} \lambda_j \nabla^2 q_j(x).$$

At (x^*, λ^*) we have:

$$\nabla L(x^*,\lambda^*) = \nabla q_0(x^*) + h_0 - \sum_{\lambda_j^* \neq 0} \lambda_j^*(\nabla q_j(x^*) + h_j) .$$

$$\nabla^2 L(x^*,\lambda^*) = \nabla^2 q_0(x^*) + \Omega'(0)D - \sum_{\lambda_j^* \neq 0} \lambda_j^* \nabla^2 q_j(x^*) .$$

If we let

$$h_0 = [\sum_{\lambda_j^* \neq 0} \lambda_j^* (\nabla q_j(x^*) + h_j) - \nabla q_0(x^*)]$$

and

$$D = [-\nabla^2 q_0(x^*) + \sum_{\lambda_j^* \neq 0} \lambda_j^* \nabla^2 q_j(x^*) + H]/\Omega'(0) \ ,$$

we will have

$$\nabla L(x^*, \lambda^*) = 0$$

and

$$\nabla^2 L(x^*, \lambda^*) = H \ .$$

So the characteristics of the Hessian of the Lagrangian at x^* can be predetermined by H. If we further select an index set $\mathbf{J}_A(x^*)$, and let

$$\alpha_j = -q_j(x^*) \quad j \in \mathbf{J}_E \cup \mathbf{J}_A(x^*)$$

$$\alpha_j \geq -q_j(x^*) \quad j \in \mathbf{J}_I \backslash \mathbf{J}_A(x^*) \ ,$$

then we will ensure the feasibility of the constraints as well as the desired activities. With proper choice of the α_j, the h_j, and of Ω and D, we can arrange that some or all of (2.2) — (2.5) will hold at x^*. To be more precise, we may

(1) Choose $\mathbf{J}_E$, $\mathbf{J}_I$, and $\mathbf{J}_A$.

(2) Choose q_0 and q_j for $j \in \mathbf{J}$.

(3) Select x^* and λ^* consistent with $\mathbf{J}_E, \mathbf{J}_I$.

(4) Choose a function Ω so that $\Omega'(0) \neq 0$.

(5) Choose vectors h_j for $j \in \mathbf{J}$ and a matrix H.

(6) Compute

$$h_0 = \sum_{\lambda_j^* \neq 0} \lambda_j^* (\nabla q_j(x^*) + h_j) - \nabla q_0(x^*)$$

and

$$D = [-\nabla^2 q_0(x^*) + \sum_{\lambda_j^* \neq 0} \lambda_j^* \nabla^2 q_j(x^*) + H]/\Omega'(0) \ .$$

(7) Choose α_0 arbitrarily, and let

$$\alpha_j = -q_j(x^*) \quad j \in \mathbf{J}_E \cup \mathbf{J}_A(x^*)$$

and

$$\alpha_j \geq -q_j(x^*) \quad j \in \mathbf{J}_I \backslash \mathbf{J}_A(x^*) \ .$$

Notes:

(a) The quantity α_0 can be selected to specify a desired value for ϕ_0 at x^*.

(b) The remaining α_j, $j \in \mathbf{J}$ provide the values of the constraint functions at x^*, $\phi_j(x^*)$. These can be chosen to ensure that $\phi_j(x^*) = 0$, $j \in \mathbf{J}_A(x^*) \cup \mathbf{J}_E$ and that $\phi_j(x^*) \geq 0$, $j \in \mathbf{J}_I$.

(c) The vectors h_j , $j \in \mathbf{J}$ can be selected arbitrarily in advance of h_0 so as to specify gradients for the constraints. Choosing these vectors to be linearly dependent for $j \in \mathbf{J}_E \cup \mathbf{J}_A(x^*)$ (assuming that any necessary constraint qualifications are satisfied for x^*) yields degenerate test cases.

(d) The λ_j^* may be chosen arbitrarily, consistent only with the inequalities of (2.3). The selection of $\lambda_j^* = 0$ for some $j \in \mathbf{J}_A(x^*)$ is often of particular difficulty for algorithms to handle properly.

(e) If the matrix H is chosen to be positive definite, then the inequalities of (2.4) and (2.5) will be satisfied automatically. A more subtle choice of H is indicated below in (g).

(f) Notes (a)-(e) provide enough flexibility in the choice of $\lambda_j, \nabla\phi_j, \nabla L, \nabla^2\phi_j, \nabla^2 L$ to specify that chosen constraints shall be active at x^*, that the gradients of these constraints shall be equal to specified vectors at x^*, and that the Lagrangian shall have a specified matrix as its Hessian at x^*. All of this is sufficient to specify that x^* shall be a minimum (or maximum or stationary point or saddle point, among other possibilities).

(g) Indefinite or ill-conditioned test problems can be obtained by letting

$$h_j = -\nabla q_j(x^*) + w_j \quad j \in \mathbf{J}_E \cup \mathbf{J}_A(x^*) \ ,$$

where the vectors w_j form an orthonormal set. Then

$$\nabla\phi_j(x^*) = w_j \quad j \in \mathbf{J}_E \cup \mathbf{J}_A(x^*) \ .$$

Let W stand for the matrix whose columns are given by the w_j; let the vectors z_i constitute a basis for the orthogonal complement of the space spanned by the w_j, and let Z represent the matrix whose columns are given by the z_i. Consequently

$$\begin{bmatrix} W^T \\ Z^T \end{bmatrix} [W \ Z] = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \ .$$

Then let

$$H = [W \ Z] \begin{bmatrix} \sigma V_1 & 0 \\ 0 & \tau V_2 \end{bmatrix} \begin{bmatrix} W^T \\ Z^T \end{bmatrix} \begin{matrix} \} \ l + |\mathbf{J}_A(x^*)| \\ \} \ n - l - |\mathbf{J}_A(x^*)| \end{matrix}$$

with positive definite matrices V_i, $i=1,2$. The matrix H will be indefinite if and only if one of σ or τ is nonpositive. For any nonzero vector $y \in \mathbf{R}^n$ we may write

$$y = y_1 + y_2 \ ,$$

where y_1 is in the space spanned by w_j and y_2 is orthogonal to this space. The condition $y^T\nabla\phi_j(x^*) = 0$, $j \in \mathbf{J}_E \cup \mathbf{J}_A(x^*)$, is equivalent to $y_1 = 0$ since $\nabla\phi_j(x^*) = w_j$. This leads to

$$y^T\nabla^2 L(x^*,\lambda^*)y = \tau y_2^T V_2 y_2 \ ,$$

which guarantees the satisfaction of the second order conditions for all values of σ, provided that $\tau > 0$. For ill-conditioned examples, V_1 and V_2 can be chosen accordingly. This scheme of generating H in a partitioned format in order to obtain ill-conditioned projected Hessians is necessary in the light of the results contained in [33].

Note that the strategy which was used provides control of the values of the ϕ_j, $j \in \mathbf{J} \cup \{0\}$ by way of the choice of the α_j. Similarly the gradients of the ϕ_j are determined by way of the choice of the h_j. The properties of ϕ_0 (or alternatively the Lagrangian) are specified by the choice of α_0, h_0, D, and Ω, so long as an "interpolating condition" on Ω, namely (3.1), holds.

This provides the fundamental outline of the way in which a single selected point $x^{(1)} = x^*$ can be made to satisfy various portions of the first and second order conditions. The next section will expand upon these ideas. In this regard, it may be seen that the ϕ_j could just as well have been defined to have the form

$$\phi_j(x) = q_j(x) + \Omega_j[\tfrac{1}{2}(x-x^*)^T D_j(x-x^*)] + \gamma_j[h_j^T(x-x^*)] + \alpha_j\delta_j(d_j^T(x-x^*)) \ .$$

So long as appropriate interpolating conditions on the functions Ω_j, γ_j, and δ_j hold, we can specify values, gradients, and Hessians via D_j, h_j, α_j, and d_j with a great deal of freedom.

4. Specifying Several Points

We will extend the idea presented in the previous section to characterize the behaviour of any finite number, s, of selected points $x^{(1)}, \ldots, x^{(s)}$.

Let

$$\begin{aligned}\phi_j(x) = q_j(x) + \sum_{i=1}^{s} \{&\alpha_{ji}\delta_{ji}[d_{ji}^T(x-x^{(i)})] \\ &+ \gamma_{ji}[h_{ji}^T(x-x^{(i)})] \\ &+ \Omega_{ji}[\tfrac{1}{2}(x-x^{(i)})^T D_{ji}(x-x^{(i)})]\},\end{aligned} \tag{4.1}$$

where $x^{(i)}$, $i=1, \ldots, s$, and $q_j(x)$, $j \in \mathbf{J} \cup \{0\}$ are given, $D_{ji} \in \mathbf{R}^{n\times n}$, d_{ji}, $h_{ji} \in \mathbf{R}^n$ and $\alpha_{ji} \in \mathbf{R}^1$ are to be chosen, and $\Omega_{ji}, \gamma_{ji}, \delta_{ji}$ are functions from $\mathbf{R}^1 \to \mathbf{R}^1$ to be determined. The functions q_j, and hence ϕ_j, are to map $\mathbf{R}^n$ into $\mathbf{R}^1$.

The gradient and the Hessian of ϕ_j are:

$$\begin{aligned}\nabla\phi_j(x) = \nabla q_j(x) + \sum_{i=1}^{s} \{&\alpha_{ji}\delta_{ji}'[d_{ji}^T(x-x^{(i)})]d_{ji} \\ &+ \gamma_{ji}'[h_{ji}^T(x-x^{(i)})]h_{ji} \\ &+ \Omega_{ji}'[\tfrac{1}{2}(x-x^{(i)})^T D_{ji}(x-x^{(i)})]D_{ji}(x-x^{(i)})\}\end{aligned}$$

$$\nabla^2\phi_j(x) = \nabla^2 q_j(x)$$
$$+ \sum_{i=1}^{t} \{\alpha_{ji}\delta_{ji}''[d_{ji}^T(x-x^{(i)})]d_{ji}d_{ji}^T + \gamma_{ji}''[h_{ji}^T(x-x^{(i)})]h_{ji}h_{ji}^T$$
$$+ \Omega_{ji}''[\tfrac{1}{2}(x-x^{(i)})^T D_{ji}(x-x^{(i)})]D_{ji}(x-x^{(i)})(x-x^{(i)})^T D_{ji}$$
$$+ \Omega_{ji}'[\tfrac{1}{2}(x-x^{(i)})^T D_{ji}(x-x^{(i)})]D_{ji}\}.$$

We introduce the following quantities:

$$\theta_{ji}(x) = d_{ji}^T(x-x^{(i)}), \quad \theta_{ji}^{(i)} = \theta_{ji}(x^{(i)}) = d_{ji}^T(x^{(i)}-x^{(i)}) \ ,$$
$$\eta_{ji}(x) = h_{ji}^T(x-x^{(i)}), \quad \eta_{ji}^{(i)} = \eta_{ji}(x^{(i)}) = h_{ji}^T(x^{(i)}-x^{(i)}) \ ,$$
$$\tau_{ji}(x) = \tfrac{1}{2}(x-x^{(i)})^T D_{ji}(x-x^{(i)}), \quad \tau_{ji}^{(i)} = \tau_{ji}(x^{(i)}). = \tfrac{1}{2}(x^{(i)}-x^{(i)})^T D_{ji}(x^{(i)}-x^{(i)}) \ .$$

Note that $\theta_{ji}^{(i)} = \eta_{ji}^{(i)} = \tau_{ji}^{(i)} = 0$.

The gradient and the Hessian of the Lagrangian are:

$$\nabla L(x,\lambda) = \nabla\phi(x) - \sum_j \lambda_j \nabla\phi_j(x)$$
$$= \nabla q_0(x)$$
$$+ \sum_{i=1}^{t} \{\alpha_{0i}\delta_{0i}'[\theta_{0i}(x)]d_{0i} + \gamma_{0i}'[\eta_{0i}(x)]h_{0i} + \Omega_{0i}'[\tau_{0i}(x)]D_{0i}(x-x^{(i)})\}$$
$$- \sum_{j\in J_E \cup J_A(x)} \lambda_j\{\nabla q_j(x) + \sum_{i=1}^{t} \{\alpha_{ji}\delta_{ji}'[\theta_{ji}(x)]d_{ji} + \gamma_{ji}'[\eta_{ji}(x)]h_{ji}$$
$$+ \Omega_{ji}'[\tau_{ji}(x)]D_{ji}(x-x^{(i)})\}\}$$

$$\nabla^2 L(x,\lambda) = \nabla^2\phi(x) - \sum_j \lambda_j \nabla^2 \phi_j(x)$$

$$= \nabla^2 q_0(x)$$

$$+ \sum_{i=1}^{s} \{\alpha_{0i}\delta_{0i}''[\theta_{0i}(x)]d_{0i}d_{0i}^T + \gamma_{0i}''[\eta_{0i}(x)]h_{0i}h_{0i}^T$$

$$+ \Omega_{0i}''[\tau_{0i}(x)]D_{0i}(x-x^{(i)})(x-x^{(i)})^T D_{0i} + \Omega_{0i}'[\tau_{0i}(x)]D_{0i}\}$$

$$- \sum_j \lambda_j \{\nabla^2 q_j(x) + \sum_{i=1}^{s} \{\alpha_{ji}\delta_{ji}''[\theta_{ji}(x)]d_{ji}d_{ji}^T$$

$$+ \gamma_{ji}''[\eta_{ji}(x)]h_{ji}h_{ji}^T$$

$$+ \Omega_{ji}''[\tau_{ji}(x)]D_{ji}(x-x^{(i)})(x-x^{(i)})^T D_{ji}$$

$$+ \Omega_{ji}'[\tau_{ji}(x)]D_{ji}\}\}.$$

The vectors d_{ji} and the functions δ_{ji} may be exploited to define $\phi_j(x^{(i)})$, for all j,t. The vectors h_{ji} and the functions γ_{ji} may be used to specify the gradient of $\phi_j(x^{(i)})$ for all j,t. (When $j=0$, the gradient of the Lagrangian or of the objective function may be specified; but not both.) The matrices D_{ji} and the functions Ω_{ji} may be defined so that the Hessian of $\phi_j(x^{(i)})$ for all j,t. is specified. (When j=0, either the Hessian of the objective function or of the Lagrangian may be specified; but not both.)

To elaborate, note how the quantities $\phi_j(x^{(i)})$, $\nabla\phi_j(x^{(i)})$, and $\nabla^2\phi_j(x^{(i)})$ depend upon the values of the functions δ_{ji}, γ_{ji}, Ω_{ji} and their first and second derivatives at the argument values $\theta_{ji}^{(i)}$, $\eta_{ji}^{(i)}$, $\tau_{ji}^{(i)}$. For a fixed j and chosen points $x^{(i)}, i = 1, \ldots, s$ suppose that the vectors d_{ji}, $i = 1, \ldots, s$, happen to have been chosen so that $\theta_{ji}^{(i)} \neq 0$ for all $i,t = 1, \ldots, s$ but $i \neq t$. Similarly, suppose h_{ji} and D_{ji}, $i = 1, \ldots, s$, have been chosen so that $\eta_{ji}^{(i)} \neq 0$ and $\tau_{ji}^{(i)} \neq 0$ for $i,t = 1, \ldots, s$ but $i \neq t$. This is a critical assumption, but one which frequently holds for small s even for randomly chosen x's, d's, h's and for nonsingular D's. Now, if a fixed index t is selected, and if the following conditions hold

$$\delta_{ji}(\theta_{ji}^{(i)}) = 0 \qquad \text{for all } i \neq t,\ i = 1, \ldots, s,$$

and

$$\delta_{ji}(\theta_{ji}^{(i)}) = \delta_{ji}(0) = 1,$$

and

$$\gamma_{ji}(\eta_{ji}^{(i)}) = 0 \qquad \text{for all } i = 1, \ldots, s(\text{including } t),$$

and

$$\Omega_{ji}(\tau_{ji}^{(i)}) = 0 \qquad \text{for all } i = 1, \ldots, s \text{ (including } t),$$

then, from (4.1), $\phi_j(x^{(t)})$ reduces to

$$\phi_j(x^{(t)}) = q_j(x^{(t)}) + \alpha_{jt} \quad .$$

Using this, the value of $\phi_j(x^{(t)})$ can be specified (for its own sake, or by way of specifying $L(x^{(t)}, \lambda^{(t)})$ for some chosen vector $\lambda^{(t)}$). The conditions listed above for δ_{ji}, γ_{ji}, Ω_{ji}, $i = 1, \ldots, s$, represent interpolatory conditions for these $3 \times s$ functions of fixed index j. Any functions constructed to satisfy these conditions will provide the means of fixing the value of ϕ_j at $x^{(t)}$.

A similar discussion involving appropriate interpolatory conditions for δ_{ji}', γ_{ji}', Ω_{ji}' will lead to the equation

$$\nabla\phi_j(x^{(t)}) = \nabla q_j(x^{(t)}) + h_{jt}$$

from which the choice of h_{jt} has served to specify $\nabla\phi_j(x^{(t)})$ (either for its own sake, or as a means of specifying $\nabla L(x^{(t)}, \lambda^{(t)})$ for a selected $\lambda^{(t)}$.) Finally, considerations involving appropriate interpolatory conditions for δ_{ji}'', γ_{ji}'', Ω_{ji}'', will serve to specify $\nabla^2\phi_j(x^{(t)})$ as

$$\nabla^2\phi_j(x^{(t)}) = \nabla^2 q_j(x^{(t)}) + D_{jt}$$

(for its own sake or as part of the process for specifying $\nabla^2 L(x^{(t)}, \lambda^{(t)})$.)

To determine various conditions at various points, then, one selects values

$$\phi_j(x^{(t)}) = q_j(x^{(t)}) + \alpha_{jt},$$

gradients

$$\nabla\phi_j(x^{(t)}) = \nabla q_j(x^{(t)}) + h_{jt}$$

and Hessians

$$\nabla^2\phi_j(x^{(t)}) = \nabla^2 q_j(x^{(t)}) + D_{jt}$$

by specifying α's, h's and D's appropriately. For a fixed t one checks that $\theta_{jt}^{(i)}$, $\eta_{jt}^{(i)}$, $\tau_{jt}^{(i)}$ are nonzero for $i \neq t$. One collects all of the interpolatory conditions together which result from the various specifications to be made, and one constructs any functions δ_{jt}, γ_{jt} and Ω_{jt} which satisfy those conditions. (We note that, if any functional value $\phi_j(x^{(t)})$ is not to be set specifically then

the corresponding appending function δ_{j_t} may be set identically to zero. This choice of δ_{j_t} here would not violate any interpolatory conditions which may arise from other specifications since those conditions only insist that the first or the second derivatives of δ_{j_t} are to be zero at other points. Likewise, γ_{j_t} or Ω_{j_t} may be set identically to zero when the corresponding gradient or Hessian is not demanded.) The result determines $\phi_0, \ldots, \phi_{t+m}$ and the characteristics of the problem (2.1). The specification of L, ∇L, $\nabla^2 L$ follows by natural extension from the above.

By way of example, the following two procedures give the flavour of specifying desired gradients and Hessians for the Lagrangian. The method of specifying values, gradients and Hessians for the ϕ_j is similar.

Procedure 4.1:

The functions $\delta_{j_t}, \gamma_{j_t}, \Omega_{j_t}, j>0$, for all i are determined elsewhere. The vectors g_{0t}, which are to be specified as $\nabla L(x^{(t)}, \lambda^{(t)})$ at selected values of $t \in \{1, \ldots, s\}$, are given.

for $t = 1$ **step** 1 **until** s **do**

if $\nabla L(x^{(t)}, \lambda^{(t)})$ is to be specified **then**

begin

{ δ_{0i}, γ_{0i}, and Ω_{0i} must satisfy

$\delta_{0i}'(\theta_{0i}^{(t)}) = 0$ for all i;

$\gamma_{0i}'(0) = 1$;

$\gamma_{0i}'(\eta_{0i}^{(t)}) = 0$ for all $i \neq t$;

$\Omega_{0i}'(\tau_{0i}^{(t)}) = 0$ for all $i \neq t$; }

$$
\begin{aligned}
h_{0t} \leftarrow [\, & g_{0t} - \nabla q_0(x^{(t)}) \\
& + \sum \lambda_j^{(t)} \{ \nabla q_j(x^{(t)}) \\
& + \sum_{i=1}^{r} \{ \alpha_{j_i} \delta_{j_i}'[\theta_{j_i}^{(t)}] d_{j_i} \\
& + \gamma_{j_i}'[\eta_{j_i}^{(t)}] h_{j_i} \\
& + \Omega_{j_i}'[\tau_{j_i}^{(t)}] D_{j_i}(x^{(t)} - x^{(t)}) \}\}] \; ;
\end{aligned}
$$

end

end do

Procedure 4.2:

The functions $\delta_{ji}, \gamma_{ji}, \Omega_{ji}, j>0$, for all i are determined elsewhere. The matrices H_{0i}, which are to be specified as $\nabla^2 L(x^{(i)}, \lambda^{(i)})$ at selected values of $t \in \{1, \ldots, s\}$, are given.

for $t = 1$ **step** 1 **until** s **do**

if $\nabla^2 L(x^{(i)}, \lambda^{(i)})$ is to be specified **then**

begin

{ δ_{0i}, γ_{0i}, and Ω_{0i} must satisfy

$\delta_{0i}''(\theta_{0i}^{(i)}) = 0$ for all i;

$\gamma_{0i}''(\eta_{0i}^{(i)}) = 0$ for all i;

$\Omega_{0i}''(\tau_{0i}^{(i)}) = 0$ for all $i \neq t$;

$\Omega_{0i}'(0) = 1$;

$\Omega_{0i}'(\tau_{0i}^{(i)}) = 0$ for all $i \neq t$; }

$$
\begin{aligned}
D_{0i} \leftarrow [H_{0i} &- \nabla^2 q_0(x^{(i)}) \\
&+ \sum \lambda_j^{(i)} \{\nabla^2 q_j(x^{(i)}) \\
&+ \sum_{j=1}^{i} \{\alpha_{ji} \delta_{ji}''[\theta_{ji}^{(i)}] d_{ji} d_{ji}^T \\
&+ \gamma_{ji}''[\eta_{ji}^{(i)}] h_{ji} h_{ji}^T \\
&+ \Omega_{ji}''[\tau_{ji}^{(i)}] D_{ji}(x^{(i)} - x^{(i)})(x^{(i)} - x^{(i)})^T D_{ji} \\
&+ \Omega_{ji}'[\tau_{ji}^{(i)}] D_{ji}\}\}] \ ;
\end{aligned}
$$

end

end do

These procedures, and their like for specifying values, gradients and Hessians for the individual ϕ_j, imply that the following (Hermite) interpolation conditions be imposed on δ_{j_t}

$\delta_{j_t}[0] = 1$

$\delta_{j_t}[\theta_{j_t}^{(i)}] = 0$ for all $i \neq t$ such that $\phi_j(x^{(i)})$ is specified.

$\delta_{j_t}'[\theta_{j_t}^{(i)}] = 0$ for all i such that the gradient at $x^{(i)}$ is specified.

$\delta_{j_t}''[\theta_{j_t}^{(i)}] = 0$ for all i such that the Hessian at $x^{(i)}$ is specified.

Likewise, for γ_{j_t} and Ω_{j_t} the corresponding conditions are:

$\gamma_{j_t}[\eta_{j_t}^{(i)}] = 0$ for all i such that $\phi_j(x^{(i)})$ is specified.

$\gamma_{j_t}'[0] = 1$

$\gamma_{j_t}'[\eta_{j_t}^{(i)}] = 0$for all $i \neq t$such that the gradient at $x^{(i)}$ is specified.

$\gamma_{j_t}''[\eta_{j_t}^{(i)}] = 0$ for all i such that the Hessian at $x^{(i)}$ is specified.

and

$\Omega_{j_t}[\tau_{j_t}^{(i)}] = 0$ for all i such that $\phi_j(x^{(i)})$ is specified.

$\Omega_{j_t}'[\tau_{j_t}^{(i)}] = 0$ for all $i \neq t$ such that the gradient or the Hessian at $x^{(i)}$ is specifiec

$\Omega_{j_t}'[0] = 1$

$\Omega_{j_t}''[\tau_{j_t}^{(i)}] = 0$ for all i such that the Hessian at $x^{(i)}$ is specified.

Notes:

(a) One should be advised that the above interpolating conditions do not constitute the only ways of achieving our ends. Some of the conditions imposed could be reduced at the expense of a more complicated presentation.

(b) If δ_{j_t}, γ_{j_t}, and Ω_{j_t} are all taken to be polynomials, then [1] provides the necessary background for fixing these functions for any suitable $\theta_{j_t}^{(i)}$, $\eta_{j_t}^{(i)}$, and $\tau_{j_t}^{(i)}$.

(c) We note that the following assumptions are necessary:

$\theta_{j_t}^{(i)} = d_{j_t}^T(x^{(i)} - x^{(t)}) \neq 0$

$$\eta_{j_t}^{(i)} = h_{j_t}^T(x^{(i)} - x^{(t)}) \neq 0 \quad \text{for all } i \neq t.$$
$$\tau_{j_t}^{(i)} = \tfrac{1}{2}(x^{(i)} - x^{(t)})D_{j_t}(x^{(i)} - x^{(t)}) \neq 0$$

The first set of these assumptions corresponding to θ_{j_t} may be satisfied automatically by choosing d_{j_t} appropriately. However, the remaining assumptions are not always guaranteed for all h_{j_t}, D_{j_t} and $x^{(i)}$, $i=1, \ldots, s$. When a failure occurs, one should merely select a new set of points $x^{(i)}$. Such failures will be rare.

5. Algorithms

In order to gain experience with the ideas described in the foregoing sections, we have prepared two experimental codes to solve the constrained nonlinear least squares problem; i.e. problem (2.1) for the special objective function

$$\underset{x}{\text{minimize}} \ \phi_0(x) = \tfrac{1}{2}F(x)^TF(x) \ ,$$

where

$$F(x) = [f_1(x), \ldots, f_p(x)] \ .$$

It is worth noting that this structure in the objective function provides the following structure to the gradient and the Hessian:

$$\nabla\phi_0(x) = J(x)^TF(x) \ ,$$

where $J(x)$ is the Jacobian of $F(x)$ (the matrix whose ith row is $\nabla f_i(x)^T$), and

$$\nabla^2\phi_0(x) = J(x)^TJ(x) + \sum_{i=1}^{p} \nabla^2 f_i(x)$$

We took two of the currently popular techniques for nonlinear programming, the successive quadratic programming technique with the watchdog modification due to Han, Powell, et al. [14, 15, 25], and the exact penalty technique due to Coleman and Conn [4, 5], and we made some modifications to these techniques to take account of the special structure of the objective function. Our modifications were along the lines of those suggested by Dennis, et. al. in [10], extended to account for constraints. Our reasons for not experimenting with (2.1) directly were twofold: (1) by treating a problem for which there is not widely available software, we were forced to write programs from scratch, which meant that we could code both techniques in a reasonably uniform manner — which we hoped would suppress artifacts arising from different coding styles, and (2) by making changes to both techniques to

account for the special nature of the problem, the blame for any poor showing of one technique over another could be laid to us for botching the job — we were interested in the mechanisms of test generation and of its utility to researchers and programmers, rather than being interested in passing judgement on specific optimization techniques. We will give a brief rundown on both techniques and indicate the modifications which we made.

A certain imbalance in the treatment will be unavoidable. The authors have worked on much of this material in close proximity to Coleman and Conn, during some of the later stages of the development of their exact penalty algorithm. It was hard to avoid becoming much more familiar with their technique than with the successive quadratic programming algorithm of Powell, *et. al.* The reader will, hopefully, make allowence for this in what follows.

The references above are to be consulted for further details about the basics and the theory — references [5, 4, 25] being the most pertinent.

For both techniques a point x^0 and an $n \times n$ positive definite matrix B^0 are needed to begin. The starting point is user-provided and the initial choice of B^0, dependent upon the technique being used, is some approximation to second derivative information. From then on, a sequence of points $\{x^k\}$ and a sequence of positive definite matrices $\{B^k\}$ $(k = 1,2,3,\ldots)$ is generated iteratively.

5.1. Successive Quadratic Programming with Watchdog

At the beginning of the k-th iteration, both x^k and B^k are known. A search direction d^k is obtained by solving the quadratic program:

$$\underset{d}{\text{minimize}} \quad d^T \nabla \phi_0(x^k) + \tfrac{1}{2} d^T B^k d$$

subject to

$$\phi_j(x^k) + d^T \nabla \phi_j(x^k) = 0 \qquad j = 1, \ldots, l$$

$$\phi_j(x^k) + d^T \nabla \phi_j(x^k) \geq 0 \qquad j = l+1, \ldots, l+m \quad .$$

The solution of this quadratic program also yields Lagrange multipliers λ^{k+1}. The search direction d^k is taken with a stepsize α^k to yield a new point $x^{k+1} = x^k + \alpha^k d^k$, and B^{k+1} is defined by

$$B^{k+1} = J(x^{k+1})^T J(x^{k+1}) + S^{k+1} \quad .$$

We have chosen S^{k+1} as a quasi-Newton approximation to:

$$\sum_{i=1}^{p} f_i(x^{k+1}) \nabla^2 f_i(x^{k+1}) \; - \; \sum_{j=1}^{l+m} \lambda_j^{k+1} \nabla^2 \phi_j(x^{k+1}) \quad .$$

The choice was made as follows:

Find $\nabla\phi_0(x^{k+1}) = J(x^{k+1})^T F(x^{k+1})$.

If $k = 0$ then set $S^0 \leftarrow 0_{n\times n}$ otherwise:

$$y^{k+1} \leftarrow [J(x^{k+1}) - J(x^k)]^T F(x^{k+1}) - \sum_{j=1}^{l+m} \lambda_j^{k+1}[\nabla\phi_j(x^{k+1}) - \nabla\phi_j(x^k)];$$

$$s^{k+1} \leftarrow x^{k+1} - x^k.$$

Use y^{k+1} and s^{k+1} in the BFGS formula (switching to DFP if using BFGS would result in a division by zero — see, e.g. [9], for a reference to these) to update S^k to S^{k+1}.

In solving the above quadratic programming problem, the techniques presented in [12] were followed:

At each step in finding the optimal value of d a working set of constraints is determined, the gradients of which, $\phi_j(x^k)$, are taken to form the columns of a matrix A.

The columns of the matrix Z are determined to form an orthonormal basis for the nullspace of A^T.

A modification to the current d is found by solving a system of equations based upon the matrix $Z^T B^k Z$.

The equation-solving process involves using the Cholesky decomposition of

$$Z^T B^k Z = LDL^T + E \quad , \tag{5.1.1}$$

where E is a diagonal matrix of small norm chosen to make $Z^T B^k Z$ positive definite. (E is a zero matrix when $Z^T B^k Z$ is already positive definite.)

After d^k is determined from the quadratic model, the stepsize α^k is chosen with the aid of two merit functions. The primary of these is

$$\psi_k(\alpha) = \Psi(x^k+\alpha d^k, \mu^k) = \phi_0(x) + \sum_{j\in J_E} \mu_j^k |\phi_j(x^k+\alpha d^k)| + \sum_{j\in J_I} \mu_j^k |\min(0, \phi_j(x^k+\alpha d^k))| \quad ,$$

where the parameters μ_j^k for each $j\in\mathbf{J}$ were updated according to:

On the first iteration:
$\mu_j^k = 2|\lambda_j^{k+1}|$.
On other iterations:
If $\mu_j^{k-1} < 1.5|\lambda_j^{k+1}|$
then $\mu_j^k = 2|\lambda_j^{k+1}|$
else $\mu_j^k = \mu_j^{k-1}$.

The secondary merit function used is merely an approximation to the Lagrangian function:

$$L_k(x^k+\alpha d^k) = \phi_0(x^k+\alpha d^k) - \sum_{j\in J} \lambda_j^{k+1}\phi_j(x^k+\alpha d^k)$$

Line searches, as needed, were performed on $\psi_k(\alpha)$ using the line search of [23] with termination set by $\eta = 0.9$. This is a very tight termination requirement, which should ensure satisfaction of the line search condition given in [25], namely

$$\Psi(x^{k+1},\mu^k) \leq \Psi(x^k,\mu^k) - \theta[\Psi(x^k,\mu^k) - \Psi_k(x^{k+1},\mu^k)] \ , \qquad (5.1.2)$$

where θ is a constant chosen from the interval $(0,\frac{1}{2})$. To ensure that the choice $\alpha = 1$ is ultimately always made, we test whether taking the point x^k+d^k as x^{k+1} will ensure that either one of the inequalities (5.1.2) or

$$L_k(x^k+d^k) < L_k(x^k)$$

is satisfied. If so, we accept x^k+d^k as x^{k+1} without carrying out a line search on $\psi_k(\alpha)$.

All other details are identical to those laid out in [25].

5.2. The Exact Penalty Technique

A single merit function is used by this technique:

$$\Psi_\epsilon(x) = \mu\phi_0(x) - \sum_{j\in J^\epsilon_{\lambda_1}(x)} \phi_j(x) + \sum_{j\in J^\epsilon_{\lambda_2}(x)} \mathrm{sgn}(\phi_j(x))\phi_j(x) \ ,$$

where for any x and $\epsilon > 0$ we define

$J^\epsilon_{\lambda_1}(x) = \{|\phi_j(x)| < \epsilon,\ j = l+1, \ldots, l+m\}$ the set of ϵ-active inequality constraints

$J^\epsilon_{\lambda_2}(x) = \{|\phi_j(x)| < \epsilon,\ j = 1, \ldots, l\}$ the set of ϵ-active

equality constraints

$\mathbf{J}^{\epsilon}_{V_1}(x) = \{\phi_j(x) < \epsilon,\ j = l+1, \ldots, l+m\}$ the set of ϵ-violated

inequality constraints

$\mathbf{J}^{\epsilon}_{V_2}(x) = \{|\phi_j(x)| > \epsilon,\ j = 1, \ldots, l\}$ the set of ϵ-violated

equality constraints.

$\mathbf{J}^{\epsilon}_{A}(x) = \mathbf{J}^{\epsilon}_{A_1}(x) \cup \mathbf{J}^{\epsilon}_{A_2}(x)$.

Implicitly with these index sets there is an approximate Lagrangian which we will denote by

$$L_{\epsilon}(x,\lambda) = \Psi_{\epsilon}(x) - \sum_{j \in \mathbf{J}^{\epsilon}_{A}(x)} \lambda_j \phi_j(x) \quad .$$

The quantity ϵ is a parameter of the method which is set positive initially and is (possibly) revised downward at certain times during the course of the algorithm. Similarly μ is set initially to 1.0, and it may be revised downward. The decisions for revision are much like those given in [4, 5]. (In particular, we will have occasion to make reference to the flowchart on page 147 of [4], where ϵ appears as ϵ_k.)

We start with x^0 and

$$B^0 = J(x^0)^T J(x^0)$$

The technique for determining B^k, x^k is divided into a global, a dropping, an intermediate, and a local mode, each characterized by the use of different quadratic subproblems and/or a different selection of step directions.

The quadratic model always has the form

$$\underset{h}{\text{minimize}} \quad \nabla\Psi_{\epsilon}(x^k)^T h + \tfrac{1}{2} h^T B^k h \qquad (5.2.1)$$

$$\text{subject to} \quad \nabla\phi_j^T(x^k) h = 0 \quad j \in \mathbf{J}^{\epsilon}_{A}(x^k) \ ,$$

but B^k approximates either $\nabla^2\Psi_{\epsilon}(x^k)$ or $\nabla^2 L_{\epsilon}(x^k, \lambda^k)$, depending upon which mode is currently in force. The modes are determined by letting

$$A^k = [\ , \ldots, \nabla\phi_j(x^k), \ldots, \ ] \quad \text{for } j \in \mathbf{J}^{\epsilon}_{A}(x^k) \ ,$$

by selecting vectors z_l to form an orthonormal basis for the nullspace of A^{kT}, by constructing the matrix Z^k with columns z_l, and then by inspecting the norm

$$|Z^T \nabla\Psi_{\epsilon}(x^k)|_2 \quad . \qquad (5.2.2)$$

We outline the decisions made and the structure of each of the modes below. Our outline is based upon a prepublication version of the exact penalty algorithm, and it is distinguished from the publication version by the presence of the intermediate mode, which serves to provide a more cautious transition from the global algorithm of [4] and its asymptotic variant, discussed in [5].

Global Mode

B^k is taken to approximate $\nabla^2\Psi_\epsilon(x^k)$ for this mode — its updating will be discussed later. Whenever (5.2.2) is greater than or equal to a tolerance τ_1, then the global mode is used, and a direction h^k is found by solving (5.2.1). A new point

$$x^{k+1} = x^k + \alpha^k h^k$$

is found by using the line search of [23] with $\eta = 0.9$ on

$$\psi_\epsilon(\alpha) = \Psi_\epsilon(x^k + \alpha^k h^k) \quad .$$

Dropping Mode

If (5.2.2) is less than τ_1, then

Lagrange multiplier estimates λ^k are obtained by solving

$$\underset{\lambda}{\text{minimize}} \; |A^k\lambda - \nabla\Psi_\epsilon(x^k)|_2 \quad , \tag{5.2.3}$$

and a test is made to see whether any $\lambda^k_{j_k}$ exists satisfying

$$\begin{aligned} &|\lambda^k_{j_k}| \in [0,+1] \quad j_k \in \mathbf{J}^k_{\lambda_1}(x^k) \quad , \\ &\text{or} \\ &|\lambda^k_{j_k}| \in [-1,+1] \quad j_k \in \mathbf{J}^k_{\lambda_2}(x^k) \quad . \end{aligned} \tag{5.2.4}$$

In this event, h^k is found by solving the underdetermined linear equations

$$\begin{aligned} \nabla\phi^T_{j_k} h &= \pm 1 \\ \nabla\phi^T_j h &= 0 \quad j \in \mathbf{J}^k_A(x^k)\backslash\{j_k\} \quad , \end{aligned} \tag{5.2.5}$$

the sign being chosen to provide descent for Ψ_ϵ. A tolerance δ is used to accept or reject h^k as providing sufficient local descent:

$$[\nabla\Psi_\epsilon(x^k) + \theta_k \nabla\phi_{j_k}(x^k)]^T h^k < \delta \quad ,$$

where $\theta_k \in \{-1, +1, 0\}$ depending upon the membership of j_k in $\mathbf{J}_E$ or $\mathbf{J}_I$ and upon the sign chosen in (5.2.5).

If no λ satisfies (5.2.4), then the intermediate mode of the technique is initiated. Otherwise a new point

$$x^{k+1} = x^k + \alpha^k h^k$$

is found as using the line search. If h^k does not pass the test of sufficient decrease, however, then ϵ, τ_1, and another tolerance τ_2 (used below) are reduced, the index sets $\mathbf{J}$ are reassessed, and the global mode is put in force.

Intermediate Mode

In the intermediate mode B^k is still taken to approximate

$$\nabla^2\Psi_\epsilon(x^k) \quad ,$$

and h^k is determined by (5.2.1). (5.2.3) is solved at the beginning of each iteration of the mode to provide values of λ_j^k, and the conditions (5.2.4) are checked. To remain in the intermediate mode, these conditions must fail to hold. (If any $\lambda_{j_k}^k$ can be found to satisfy (5.2.4), then the dropping mode is put into force.) Assuming that the intermediate mode is permitted to continue, h^k is determined from (5.2.1), and a line search is made on $\psi_\epsilon(\alpha) = \Psi_\epsilon(x^k + \alpha h^k)$. This yields the point $x^k + \alpha^k h^k$. The vector

$$\Phi(x^k + \alpha^k h^k) = [\cdots, \phi_j(x^k + \alpha^k h^k), \cdots]^T \quad , \quad j \in \mathbf{J}_A^\epsilon(x^k)$$

is formed; a direction v^k is found by solving

$$A^{kT} v = -\Phi(x^k + \alpha^k h^k) \quad , \tag{5.2.6}$$

and the point

$$x^k + \alpha^k h^k + v^k$$

is considered. If a tolerance test indicates that

$$\Psi_\epsilon(x^k + \alpha^k h^k + v^k) \text{ is sufficiently smaller than } \Psi_\epsilon(x^k) \quad ,$$

then

$$x^{k+1} = x^k + \alpha^k h^k + v^k \quad ,$$

otherwise

$$x^{k+1} = x^k + \alpha^k h^k \quad ;$$

ϵ, τ_1, and τ_2

are reduced, and the global mode is put in force.

Local Mode

If the norm (5.2.2) is less than a tolerance τ_2 ($\tau_2 < \tau_1$), and if $x^k + \alpha^k h^k + v^k$ has been accepted as x^{k+1} on the intermediate-mode iteration step last taken, then the local mode will be in force. In this mode B^k is maintained as a quasi-Newton approximation to

$$\nabla^2 L_e(x^k, \lambda^k) \quad ,$$

and a direction h^k is determined from (5.2.1). The quantity $\alpha = \alpha^k$ is set to 1 without a line search, and the vector v^k is determined from (5.2.6) with this version of B^k. The point

$$x^{k+1} = x^k + h^k + v^k$$

is accepted, provided a tolerance test indicates that $\Psi_e(x^k + h^k + v^k)$ shows sufficient decrease over $\Psi_e(x^k)$. If this point is not accepted, then the intermediate mode is put into force after

ϵ, τ_1, and τ_2

have been reduced.

Updating

The quasi-Newton updating for

$$B^k = \mu J(x^k)^T J(x^k) + S^k$$

must take account of the fact that

$$S^k \approx \nabla^2 \Psi_e(x^k) - \mu J(x^k)^T J(x^k)$$

in the global and intermediate mode, and that

$$S^k \approx \nabla^2 L_e(x^k, \lambda^k) - \mu J(x^k)^T J(x^k)$$

in the local mode. This is attempted by setting

$$y^{k+1} \leftarrow \mu[J(x^{k+1})-J(x^k)]^T F(x^{k+1}) - \sum_{j\in J^g_{\bar{V}_1}(x^{k+1})} [\nabla\phi_j(x^{k+1})-\nabla\phi_j(x^k)] \quad (5.2.7)$$

$$+ \sum_{j\in J^g_{\bar{V}_2}(x^{k+1})} \operatorname{sgn}(\phi_j(x^{k+1}))[\nabla\phi_j(x^{k+1})-\nabla\phi_j(x^k)]$$

$$+ \textbf{ if } \{\text{local mode}\} \textbf{ then}$$

$$\sum_{j\in J^g_A(x^{k+1})} \lambda_j^{k+1}[\nabla\phi_j(x^{k+1})-\nabla\phi_j(x^k)]$$

$$\textbf{else } 0 \ ,$$

and

$$s^{k+1} \leftarrow x^{k+1} - x^k \ .$$

The vectors y^{k+1} and s^{k+1} are used in the BFGS formula to update S^k to S^{k+1} (switching to DFP if division by zero is to be avoided).

In any event, the solution of (5.2.1), for B^k obtained in this fashion, is accomplished in all modes (global, intermediate, and local) via the LDL^T decomposition described in (5.1.1).

Comparisons

The above has been a superficial, verbal description of the algorithm given in flowchart form on page 147 of [4] provided certain changes are introduced (the foremost of which being to correct the chart by introducing μ into those boxes from which it is missing).

1. The quantity Λ_k in the chart has been split into τ_1 and τ_2 in the description, and the dependence upon k has been suppressed. Likewise ϵ in the description corresponds to ϵ_k in the chart.
2. The box at the top of the chart containing

$$k \leftarrow k+1$$
$$\text{Identify } I_A^{\epsilon_k}$$

should now read

$k \leftarrow k+1$
Identify I_A^*
NEWT ← **false**

3. The box containing

$$h^k \leftarrow -Z_k(Z_k^T B_k Z_k)^{-1} Z_k^T \bar{\nabla} p$$
Solve $A_k^T v^k \approx -\Phi(x^k + h^k)$

should be split into several parts. To begin, there should be a box containing

$$\alpha_k \leftarrow 1$$
$$h^k \leftarrow -Z_k(Z_k^T B_k Z_k)^{-1} Z_k^T \bar{\nabla} p$$

Following this there should be a diamond, asking

$$|Z_k^T \bar{\nabla} p| > \tau_1$$
and
NEWT = **true**
?

If the answer is "YES", then downward flow should continue into a box containing

$$\text{Solve } A_k^T v^k \approx -\Phi(x^k + h^k) \quad ,$$

and further action is as shown on page 147. If the answer is "NO", then flow should be directed to the right, into a box with

Determine α_k via
Algorithm 1

Following this should be a box with

$$\text{Solve } A_k^T v^k \approx -\Phi(x^k + \alpha_k h^k)$$

and a diamond with the question

$$p(x^k + \alpha_k h^k + v^k) < p(x^k + \alpha_k h^k) - \beta \ ?$$

(where β is some fixed, positive tolerance which determines acceptable decrease). If the answer is "YES", then flow proceeds to the bottom, left box in the chart, which now reads

$$\text{NEWT} \leftarrow \textbf{true}$$
$$x^{k+1} \leftarrow x^k + \alpha_k h^k + v^k$$
$$k \leftarrow k+1$$

If the answer is "NO", then flow proceeds to a box which contains

$$x^{k+1} \leftarrow x^k + \alpha_k h^k$$
$$\text{NEWT} \leftarrow \textbf{false}$$

and this box exits into the one at the lower right of the flowchart, which adjusts ϵ, τ_1, and τ_2:

$$\epsilon \leftarrow \epsilon/2$$
$$\tau_1 \leftarrow \tau_1/2$$
$$\tau_2 \leftarrow \tau_2/2$$
$$\text{Identify } I_A^q$$

6. Example Results

The nonlinear programming problems to be considered here have the form

$$\underset{x}{\text{minimize}} \quad \tfrac{1}{2}F(x)^T F(x) + \Omega_0(\tfrac{1}{2}(x-x^*)^T D(x-x^*)) + \gamma_0((x-x^*)^T h_0)$$

subject to

$$\phi_j(x) = q_j(x) + h_j^T(x-x^*) + \alpha_j = 0 \qquad j = 1, \ldots, l$$

$$\phi_j(x) = q_j(x) + h_j^T(x-x^*) + \alpha_j \geq 0 \qquad j = l+1, \ldots, l+m$$

where $F(x) = [f_1(x), \ldots, f_p(x)]^T$, with

$$f_i(x) = \sum_{l=1}^{n} (x_l^2 x_l) - i \qquad i = 1, \ldots, p \quad ,$$

where

$$q_j(x) = \sum_{l=1}^{n} (x_l^2 x_l) - j \qquad j = 1, \ldots, l+m \quad ,$$

and where

$$\Omega_0(t) = \gamma_0(t) = \tfrac{1}{2}(t+1)^2 \quad \text{for } t \in \mathbf{R}$$

Note that this is consistent with the choice

$$\alpha_0 = 0$$

$$\gamma_j = \text{identity}$$

$$\delta_j = 1$$

$$\Omega_j = 0 \ .$$

We have arranged the form of Ω_0 and γ_0 so that the objective function will be a sum of squares.

In all cases we assigned values to the optimizer and its associated Lagrange multipliers by using the portable random number generator given in [32]. The components of x^* and the Lagrange multipliers of the active equality constraints λ^* were chosen between -1.0 and 1.0, while the Lagrange multipliers of the active inequality constraints were chosen between 0.0 and 1.0. We had

$n = 5$ - number of variables
$p = 5$ - number of components in $F(x)$
$l = 2$ - number of equality constraints
$m = 3$ - number of inequality constraints
$\nu = 2$ - number of active inequality constraints

and for all problems

$$\text{Optimizer } x^* = \begin{pmatrix} 0.30467 \\ 0.62076 \\ -0.80999 \\ 0.41441 \\ 0.95425 \end{pmatrix}$$

Active constraints set = { 1,2,3,4 }

$$\text{Lagrange multipliers } \lambda^* = \begin{pmatrix} 0.12629 \\ 0.59378 \\ 0.31597 \\ 0.58508 \end{pmatrix}$$

The Lagrangian Hessian was set to be

$$H = \begin{bmatrix} \sigma U_1^T U_1 & 0 \\ 0 & U_2^T U_2 \end{bmatrix} ,$$

where U_1 was a $(\nu+l)\times(\nu+l)$ upper triangular matrix. Except for $U_1[1,1]$, which was set to 2, all elements of U_1 were generated by using a random number generator and were in the range -1.0 to 1.0. U_2 was also an upper

triangular matrix with elements all generated between -1.0 and 1.0. The constant σ determined whether H was positive definite or indefinite, and $\sigma = 1, -1$, and -10 were used. To obtain nondegenerate test problems, the gradient of each constraint ϕ_j at x^* was set to e_j (the elementary column vector with component j equal to one and all others zero). To obtain degenerate problems, all save the fourth constraint gradient were set this way, and the gradient of the fourth active constraint was chosen to be linearly dependent on the gradients of the first three active constraints. (We trusted to luck on the constraint qualifications.) The constants α_j were set so that $\phi_j(x^*) = 0$ for the active constraints, and $\phi_j(x^*) = 1$ for the inactive constraints.

Implementation:

All codes were written in ANSII FORTRAN, as verified by the PFORT verifier [27], and were run on the Honeywell 66/60 of the Mathematics Faculty Computing Facility at the University of Waterloo. The generation of random numbers was based upon the technique of [32], modified for double precision. The basic linear algebra computations (e.g. dot products) were carried out using the Basic Linear Algebra Subroutines of [18], and the more advanced linear algebra (e.g. forming QR factorizations and finding the LDL^T factorization of a — possibly nonnegative definite — symmetric matrix) is carried out by using or modifying programs from the NPL optimization library [13]. The quadratic programming code needed for the technique of [25] was also obtained from this library, which was made available through the courtesy of P. E. Gill and W. Murray. The line search routine which was used in all codes comes from reference [23] and was kindly provided by Michael Overton. All random numbers were produced by [32] as modified for double precision.

Employment

We illustrate the use of these test problems in two typical ways, firstly in the "shotgun" mode of software testing, in which the problems are dumped into the codes and the results are summarized in tabular or statistical format, secondly in a "scalpel" mode, in which selected problems are followed, step-by-step, through an algorithm see what insights can be gained. For the former mode, the computational results are given in tables which follow. Unless otherwise indicated, all tests were terminated if more than 50 iterations were taken, in the interests of saving our computing budget. For the latter mode, we include summary remarks after each table indicating a few of the interesting features of the execution. Again, we remind the reader that our remarks reflect studies of the techniques of [4, 5, 25] as subjected to our interpretation, for a problem of special structure, applying quasi-Newton

updates of our choice, and suffering (possibly) under our blind spots in rendering these techniques into computer code.

Table Headings

EP - exact penalty method
SQP - successive quadratic programming with watchdog technique
NI - number of iterations
SP - starting point:
starting point 1 - x with all component equal to 1.0
starting point 2 - x with all component equal to 10.0
starting point 3 - x with all component equal to 100.0.
FE - number of objective function evaluations
(each evaluation of F was counted as $p+2$)
CE - number of constraint evaluations
GFE - number of gradient evaluations in the objective function
(each Jacobian evaluation was counted as $p+2$)
GCE - number of gradient evaluations in the constraints
R - result:
C - Constructed optimizer found
A - Another point of termination
F - Failure, code aborted
M - Maximum iteration count reached

Tables

Non-degeneracy with positive definite H ($\sigma = 1.0$)

TABLE 1

SP	EP						SQP					
	NI	FE	CE	GFE	GCE	R	NI	FE	CE	GFE	GCE	R
1	22	245	175	210	169	C	19	147	105	105	103	C
2	35	490	350	435	375	C	10	98	70	70	65	F
3	38	497	355	420	448	C	47	434	310	310	298	C

Remarks:

(1) In SQP with starting point 1 the first 4 penalty parameters were determined by the Lagrange multipliers of the first quadratic subproblem, and the last penalty parameter was determined by a multiplier of the second quadratic subproblem. The penalty parameters ranged from 51

to 350 and remained unchanged throughout iterations 3 - 19.

(2) In SQP with starting point 2, the penalty parameters were set between 16750 and 89016 by the first quadratic subproblem. At iteration 6, a quadratic-programming subproblem was encountered with Lagrange multipliers of the order of 10^9. This forced the penalty parameters to become large as well. On the next two iterations, multipliers became 10^{23} and 10^{36} respectively. The line search became unable to find lower merit function values with the search directions produced, and the code was aborted.

(3) In SQP, with starting point 3, the penalty parameters had been set to numbers of magnitude 10^6 by iteration 3. These values held throughout the subsequent iterations without change.

(4) It is a general observation that the first few quadratic programs encountered by SQP for all of our starting points and for all of our generated problems were associated with large Lagrange multipliers. These, in turn, set the values of the penalty parameters quite large, and subsequent quadratic programs, with their much more reasonably sized multipliers, had no influence on the magnitude of the parameters. A revision of the code with some sort of "restart" provision is needed to provide for a periodic (or dynamic) reassessment of the penalty parameters to account for cases in which Lagrange multipliers show significant decreases. This would not necessarily have helped in (2) above, since multiplier growth was generally upward in that problem, nor would it have prevented the failures recorded in tables below, for which the same growth was true. But it might have made a significant difference in the following cases in which SQP was not able to reach termination within 50 iterations.

(5) Each of the three executions of the EP method was characterized by an initial number of steps in which all constraints were deemed active, causing (very large) Lagrange multipliers to be computed. These generated "dropping directions", (5.2.5). There followed an intermediate number of global iterations in which steps were obtained from (5.2.1), during which $|Z^T\nabla\Psi_\epsilon|$ decreased almost monotonically. These steps gathered the activities which represented the optimal constraint manifold, and ended when the intermediate mode was entered. After several intermediate mode iterations, the local mode was begun, and convergence to the constructed optimizer decidedly appeared to be superlinear. For starting point 2, for example, the breakdown was: 7 iterations of dropping steps, 16 of global steps, 3 intermediate steps, and 9 local steps. The components of x^k agreed with those at termination in the final 5 steps respectively to 3, 4, 7, 12, and >15 digits.

Non-degeneracy with mildly indefinite H ($\sigma = -1.0$)

TABLE 2

SP	EP						SQP					
	NI	FE	CE	GFE	GCE	R	NI	FE	CE	GFE	GCE	R
1	23	252	180	205	159	C	22	182	130	130	126	C
2	32	455	325	385	334	C	10	98	70	70	65	F
3	44	518	370	395	347	A	50	672	480	480	464	M

Remark:

(1) The SQP method failed in the same manner as for starting point 2 in the previous problem — quadratic programs were encountered with large Lagrange multipliers.

(2) For starting point 3, the EP method terminated on a point (not the constructed optimizer) which satisfied the first order necessary conditions. The code does not check second order conditions.

Non-degeneracy with strongly indefinite H ($\sigma = -10.0$)

TABLE 3

SP	EP						SQP					
	NI	FE	CE	GFE	GCE	R	NI	FE	CE	GFE	GCE	R
1	50	1015	830	675	634	A	19	140	100	100	99	C
2	50	868	625	715	646	M	10	98	70	70	65	F
3	50	798	590	645	577	M	33	238	170	170	169	C

Remarks:

(1) The EP code converged to a stationary point (a first order point which was infeasible for the given constraints) on iteration 30. Progress to this stage involved an initial mixture of global and dropping steps, an attempted intermediate step followed by several successful intermediate steps, and a final number of local steps. The stationary point had constraints 2 and 4 active (1, 2, 3, and 4 are active at the constructed optimizer), and was infeasible in constraint 1. The penalty parameter μ was reduced to an eighth of its value, and the remaining 20 iterations brought convergence to a point which satisfied the first order conditions. This point was $[0.299, 0.610, -0.740, 0.407, 0.87]$, which is close to the constructed optimizer, but it has only constraints 1, 2, and 4 active

(values $\approx 10^{-20}$). Constraint 3, which is zero at the constructed optimizer, has the value 0.0401 at this point. There is a good chance that the projected Hessian of the Lagrangian — by the nature of the Hessian's construction, by the proximity of this point to the constructed optimizer, and by the fact that the gradient of constraint 3 is not included in the projection — is indefinite. This should be examined. This points to a possible limitation in the EP code (using (5.2.7) to update the matrix B of (5.2.1) and using the Cholesky decomposition indicated by (5.1.1)). Namely, second order information relating to negative curvature is being ignored.

(2) The EP method behaved much the same way on starting points 2 and 3, but lagged enough behind its progress on starting point 1 that it was cut off at iteration 50 just on or after a reduction in μ.

(3) SQP failure was as before.

Degeneracy with positive definite H ($\sigma = 1.0$)

TABLE 4

SP	EP						SQP					
	NI	FE	CE	GFE	GCE	R	NI	FE	CE	GFE	GCE	R
1	33	392	280	285	237	C	50	658	470	470	445	M
2	50	791	656	615	555	M	50	728	520	520	516	M
3	49	588	420	425	455	C	22	1519	1085	1085	1080	F

Remarks:

(1) With starting points 1 and 3, EP converged to the optimizer. The Lagrange multipliers at the termination point were in agreement with the first order conditions, but they were different in value from those assigned due to the linear dependence of the constraint gradients.

(2) For starting point 2, EP attained a stationary point, reduced μ, and was making normal progress when iteration 50 was reached.

(3) The failures of SQP were as above.

Degeneracy with mildly indefinite H ($\sigma = -1.0$)

TABLE 5

SP	EP						SQP					
	NI	FE	CE	GFE	GCE	R	NI	FE	CE	GFE	GCE	R
1	50	623	445	485	391	M	50	1379	985	985	926	M
2	50	1645	1195	1245	1137	M	49	1050	750	750	746	F
3	50	623	445	565	456	M	29	315	225	225	225	C

Remarks:

(1) For starting point 1, the behaviour noted above was to be seen. A stationary point was reached, μ was reduced, and normal progress appeared to continue as iteration 50 was reached. To verify the impression that progress was normal, a second run was made with 100 iterations allowed. The method terminated at the constructed optimum after 95 iterations.

(2) For starting point 2, the EP code stalls. The tolerance used to judge sufficient decrease in the "constraint dropping direction", (5.2.5), was set at 0.02 for all of the problems. This appears to be too small for this one case. The method is able to make no progress in dropping a constraint, yet is not prompted to reduce its tolerance, ϵ, used for judging activity by failing the "sufficient decrease test". *A priori* selection of tolerances, and associated issues of problem scaling, are perennial items of concern in nonlinear programming codes.

(3) For starting point 3, the EP code attained a stationary point around iteration 50. It was cut off before a reduction in μ could take place.

(4) The SQP method again developed large penalty parameters and ceased making progress.

Summary:

We make note of the fact that the problems were easy to generate, that the generation process was flexible enough to handle a special request on the form of the objective function (i.e. that it be a sum of squares), and that useful information pointing to the areas which need further study was readily obtained.

Acknowledgements:

The authors wish to thank A. R. Conn, who has given freely of his suggestions and encouragement. Thanks are also due to R. H. Byrd and A. L. Goldman, who undertook careful readings of the thesis from which much of this material is derived, and to W. Murray, P. E. Gill, and M. L. Overton

for providing the use of various of their software routines. Finally, the authors are grateful to J. T. Hon for generating the example problems reported here, running the algorithms, and creating the tables of this section.

7. References

[1] A. Björck and T. Elfving (1973), Algorithms for Confluent Vandermonde Systems, *Numerische Mathematik* **21**, 130-137.

[2] A. Charnes, W.M. Raike, J.D. Stutz, and A.S. Walters (1974), On Generation of Test Problems for Linear Programming Codes, *CACM* **17**(10), 583-586.

[3] A.R. Coleville (1970), A Comparative Study on Nonlinear Programming Codes, *Proceedings of the Princeton Symposium on Mathematical Programming*, Princeton University Press, Princeton, New Jersey, H. W. Kuhn (ed.).

[4] A.R. Conn and T.F. Coleman (1982), Nonlinear Programming via an Exact Penalty Function: Global Analysis, *Mathematical Programming* **24**, North-Holland, 137-161.

[5] A.R. Conn and T.F. Coleman (1982), Nonlinear Programming via an Exact Penalty Function: Asymptotic Analysis, *Mathematical Programming* **24**, North-Holland, 123-136.

[6] H. Crowder, R. S. Dembo, and J. M. Mulvey (1978), Reporting Computational Experiments in Mathematical Programming, *Mathematical Programming* **15**, North-Holland, 316-329.

[7] H. Crowder, R. S. Dembo, and J. M. Mulvey (1979), On Reporting Computational Experiments with Mathematical Software, *ACM Transactions on Mathematical Software*, **5**(2), June, 193-203.

[8] R. S. Dembo (1976), A Set of Geometric Programming Test Problems and their Solutions, *Mathematical Programming* **10**, North-Holland, 192-213.

[9] J. E. Dennis, Jr. and J. J. More (1977), Quasi-Newton Methods: Motivation and Theory, *SIAM Review* **19**, 46-89.

[10] J. E. Dennis, Jr., D. M. Gay, and R. E. Welsch (1981), An Adaptive Nonlinear Least-Squares Algorithm, *ACM Trans. on Mathematical Software* **7**, 348-383.

[11] A.V. Fiacco and G.P. McCormick (1968), *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, John Wiley and

Sons, New York.

[12] P. E. Gill and W. Murray (1978), Numerically Stable Methods for Quadratic Programming, *Mathematical Programming* **14**, North-Holland, 349-372.

[13] P.E. Gill, W. Murray, and S. Picken (1978), The NPL Numerical Optimization Software Library, Division of Numerical Analysis and Computer Science, National Physical Laboratory, Teddington, Middlesex, TW11 OLW, England.

[14] S-P. Han (1976), Superlinearly Convergent Variable Metric Algorithms for General Nonlinear Programming Problems, *Mathematical Programming* **11**, North-Holland, 263-282.

[15] S-P. Han (1977), A Globally Convergent Method for Nonlinear Programming, *Journal of Optimization Theory and Applications* **22**(3), July.

[16] D. M. Himmelblau (1972), *Applied Nonlinear Programming*, McGraw-Hill, New York, New York.

[17] K. L. Hoffman and D. R. Shier (1980), A Test Problem Generator for Discrete Linear L1 Approximation Problems, *ACM Trans. Math Software* **6**, 587-593.

[18] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh (1979), Basic Linear Algebra Subprograms for FORTRAN Usage, *ACM Trans. on Math. Software* **5**, 308-325.

[19] J. H. May and R. L. Smith (1980), Random Polytopes: Their Definition, Generation and Aggregate Properties, 80-6, Department of Industrial and Operations Engineering, University of Michigan , Ann Arbor, Michigan, USA 48106, 31 pages.

[20] W. M. Michaels and R. P. O'Neill (1980), A Mathematical Program Generator MPGENR, *A.C.M. Trans. on Math. Software* **6**, March, 31-44.

[21] J.J. More (1978), Implementation and Testing of Optimization Software, , *IFIP WG 2.5 Working Conference on Performance Evaluation of Numerical Software*, December, Baden, Austria.

[22] J.J. More, B.S. Garbow, and K.E. Hillstrom (1981), Algorithm 566: FORTRAN Subroutines for Testing Unconstrained Optimization Software, *ACM Trans. on Math. Software*, vol. 7, 136-140.

[23] W. Murray and M.L. Overton (1978), Steplength Algorithms for Minimizing a Class of Nondifferentiable Functions, STAN-CS-78-679, Computer Science Department, Stanford University, Stan-

ford, California, November.

[24] W. Murray and M. H. Wright (1978), Projected Lagrangian Methods Based on the Trajectories of Penalty and Barrier Functions, SOL 78-23, Operations Research Department, Systems Optimization Laboratory, Stanford University, Stanford, California, USA 94305, 67 pages.

[25] M.J.D. Powell, R.M. Chamberlain, C. Lemarechal, and H.C. Pedersen (1979), The Watchdog Technique for Forcing Convergence in Algorithms for Constrained Optimization, *Tenth International Symposium on Mathematical Progamming*, Montreal, Canada, August.

[26] J.B. Rosen and S. Suzuki (1965), Construction of Nonlinear Programming Test Problems, *CACM* **8**(2), 113.

[27] B. G. Ryder (1974), The PFORT Verifier, *Software Practice and Experience* **4**, 359-377.

[28] R.W.H. Sargent (1982), Recursive Quadratic Programming Algorithms and their Convergence Properties, *Numerical Analysis, Proceedings, Cocoyoc, Mexico 1981*, J.P. Hennert (ed.), Springer-Verlag [Lecture Notes Mathematics #909].

[29] K. Schittkowski (), A Numerical Comparison of 13 Nonlinear Programming Codes with Randomly Generated Test Problems, *Numerical Optimization of Dynamic Systems*, L.C.W. Dixon and G.P. Szego (eds.), North-Holland Publishing Company. [to appear].

[30] K. Schittkowski (1979), Randomly Generated Nonlinear Programming Test Problems, *9-th IFIP Conference on Optimization Techniques*, Warszawa.

[31] K. Schittkowski and W. Hock (1981), Test Examples for Nonlinear Programming Codes, Springer-Verlag [Lecture Notes in Economic and Mathematical Systems #187].

[32] L. Schrage (1979), *A More Portable FORTRAN Random Number Generator*, vol. 5, .

[33] G. W. Stewart (1981), Constrained Definite Hessians Tend to Be Well Conditioned, *Mathematical Programming* **21**, North-Holland, 235-238.

[34] J. Telgen and W. B. vanDam (1979), Randomly Generated Polytopes for Testing Mathematical Programming Algorithms, 7929/0, Economics Institut, Erasmus University, HOLLAND, 14 pages.