TECHNICAL RESEARCH REPORT

S Y S T E M S
R E S E A R C H
C E N T E R

Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

User's Guide for FSQP Version 3.0c: A FORTRAN Code for Solving Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality and Linear Constraints

by J.L. Zhou and A.L. Tits

TR 92-107

User's Guide for FSQP Version 3.0c: A FORTRAN Code for Solving Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality and Linear Constraints[1]

Jian L. Zhou and André L. Tits

Electrical Engineering Department
and
Systems Research Center
University of Maryland, College Park, MD 20742

Abstract

FSQP 3.0c is a set of FORTRAN subroutines for the minimization of the maximum of a set of smooth objective functions (possibly a single one) subject to general smooth constraints. If the initial guess provided by the user is infeasible for some inequality constraint or some linear equality constraint, FSQP first generates a feasible point for these constraints; subsequently the successive iterates generated by FSQP all satisfy these constraints. Nonlinear equality constraints are turned into inequality constraints (to be satisfied by all iterates) and the maximum of the objective functions is replaced by an exact penalty function which penalizes nonlinear equality constraint violations only. The user has the option of either requiring that the (modified) objective function decrease at each iteration after feasibility for nonlinear inequality and linear constraints has been reached (monotone line search), or requiring a decrease within at most four iterations (nonmonotone line search). He/She must provide subroutines that define the objective functions and constraint functions and may either provide subroutines to compute the gradients of these functions or require that FSQP estimate them by forward finite differences.

FSQP 3.0c implements two algorithms based on Sequential Quadratic Programming (SQP), modified so as to generate feasible iterates. In the first one (monotone line search), a certain Armijo type arc search is used with the property that the step of one is eventually accepted, a requirement for superlinear convergence. In the second one the same effect is achieved by means of a (nonmonotone) search along a straight line. The merit function used in both searches is the maximum of the objective functions if there is no nonlinear equality constraint.

[1]This research was supported in part by NSF's Engineering Research Centers Program No. NSFD-CDR-88-03012, by NSF grant No. DMC-88-15996 and by a grant from the Westinghouse Corporation.

Conditions for External Use

1. The FSQP routines may not be distributed to third parties. Interested parties should contact the authors directly.
2. If modifications are performed on the routines, these modifications will be communicated to the authors. The modified routines will remain the sole property of the authors.
3. Due acknowledgment must be made of the use of the FSQP routines in research reports or publications. A copy of such reports or publications should be forwarded to the authors.
4. The FSQP routines may not be used in industrial production, unless this has been agreed upon with the authors in writing.

User's Guide for FSQP Version 3.0c (Released September 1992)

Enquiries should be directed to

Prof. André L. Tits
Electrical Engineering Dept.
and Systems Research Center
University of Maryland
College Park, Md 20742
U. S. A.

Phone : 301-405-3669
Fax : 301-405-6707
E-mail : andre@src.umd.edu

Contents

1 Introduction **2**

2 Description of the Algorithms **3**

3 Specification of Subroutine FSQPD 3.0b **11**

4 User-Accessible Stopping Criterion **15**

5 Description of the Output **16**

6 User-Supplied Subroutines **18**

6.1 Subroutine obj 18

6.2 Subroutine constr 19

6.3 Subroutine gradob 20

6.4 Subroutine gradcn 20

7 Organization of FSQPD and Main Subroutines **21**

7.1 Main Subroutines 21

7.2 Other Subroutines 22

7.3 Reserved Common Blocks 22

8 Examples **23**

9 Results for Test Problems **34**

10 Limitations **36**

1 Introduction

FSQP (Feasible Sequential Quadratic Programming) 3.0c is a set of FORTRAN subroutines for the minimization of the maximum of a set of smooth objective functions (possibly a single one) subject to nonlinear equality and inequality constraints, linear equality and inequality constraints, and simple bounds on the variables. Specifically, FSQP tackles optimization problems of the form

$$(P) \qquad \min \max_{i\in I^f}\{f_i(x)\} \quad \text{s.t.} \quad x\in X$$

where $I^f = \{1,\ldots,n_f\}$ and X is the set of point $x \in R^n$ satisfying

$$\begin{array}{l}
bl \le x \le bu \\
g_j(x) \le 0, \quad j=1,\ldots,n_i \\
g_j(x) \equiv \langle c_{j-n_i}, x\rangle - d_{j-n_i} \le 0, \quad j = n_i+1,\ldots,t_i \\
h_j(x) = 0, \quad j=1,\ldots,n_e \\
h_j(x) \equiv \langle a_{j-n_e}, x\rangle - b_{j-n_e} = 0, \quad j = n_e+1,\ldots,t_e
\end{array}$$

with $bl \in R^n$; $bu \in R^n$; $f_i : R^n \to R$, $i=1,\ldots,n_f$ smooth; $g_j : R^n \to R$, $j = 1,\ldots,n_i$ nonlinear and smooth; $c_j \in R^n$, $d_j \in R$, $j=1,\ldots,t_i-n_i$; $h_j : R^n \to R$, $j = 1,\ldots,n_e$ nonlinear and smooth; $a_j \in R^n$, $b_j \in R$, $j=1,\ldots,t_e-n_e$.

If the initial guess provided by the user is infeasible for nonlinear inequality constraints and linear constraints, FSQP first generates a point satisfying all these constraints by iterating on the problem of minimizing the maximum of these constraints. Then, using Mayne-Polak's scheme [1], nonlinear equality constraints are turned into inequality constraints[2]

$$h_j(x) \le 0, \quad j=1,\ldots,n_e$$

and the original objective function $\max_{i\in I^f}\{f_i(x)\}$ is replaced by the modified objective function

$$f_m(x,p) = \max_{i\in I^f}\{f_i(x)\} - \sum_{j=1}^{n_e} p_j h_j(x),$$

where p_j, $j=1,\ldots,n_e$, are positive penalty parameters and are iteratively adjusted. The resulting optimization problem therefore involves only linear constraints and nonlinear inequality constraints. Subsequently, the successive iterates generated by FSQP all satisfy these constraints. The user has the option of either requiring that the exact penalty function (the maximum value of the objective functions if without nonlinear equality constraints) decrease at each iteration after feasibility for original nonlinear inequality and linear constraints has been reached, or requiring a decrease within at most three iterations. He/She

[2]For every j for which $h_j(x_0) > 0$ (x_0 is the initial point), "$h_j(x) = 0$" is first replaced by "$-h_j(x) = 0$" and $-h_j$ is renamed h_j.

must provide subroutines that define the objective functions and constraint functions and may either provide subroutines to compute the gradients of these functions or require that FSQP estimate them by forward finite differences.

Thus, FSQP 3.0c solves the original problem with nonlinear equality constraints by solving a modified optimization problem with only linear constraints and nonlinear inequality constraints. For the transformed problem, it implements algorithms that are described and analyzed in [2], [3] and [4], with some additional refinements. These algorithms are based on a Sequential Quadratic Programming (SQP) iteration, modified so as to generate feasible iterates. The merit function is the objective function. An Armijo-type line search is used to generate an initial feasible point when required. After obtaining feasibility, either (i) an Armijo-type line search may be used, yielding a monotone decrease of the objective function at each iteration [2]; or (ii) a nonmonotone line search (inspired from [5] and analyzed in [3] and [4] in this context) may be selected, forcing a decrease of the objective function within at most four iterations. In the monotone line search scheme, the SQP direction is first "tilted" if nonlinear constraints are present to yield a feasible direction, then possibly "bent" to ensure that close to a solution the step of one is accepted, a requirement for superlinear convergence. The nonmonotone line search scheme achieves superlinear convergence with no bending of the search direction, thus avoiding function evaluations at auxiliary points and subsequent solution of an additional quadratic program. After turning nonlinear equality constraints into inequality constraints, these algorithms are used directly to solve the modified problems. Certain procedures (see, e.g., [6]) are adopted to obtain proper values of p_j's in order to ensure that a solution of the modified problem is also a solution of the original problem, as described below.

For the solution of the quadratic programming subproblems, FSQP 3.0c is set up to call QLD [7] which is provided with the FSQP distribution for the user's convenience.

2 Description of the Algorithms

The algorithms described and analyzed in [2], [3] and [4] are as follows. Given a feasible iterate x, the basic SQP direction d^0 is first computed by solving a standard quadratic program using a positive definite estimate H of the Hessian of the Lagrangian. d^0 is a direction of descent for the objective function; it is almost feasible in the sense that it is at worst tangent to the feasible set if there are nonlinear constraints and it is feasible otherwise.

In [2], an essentially arbitrary feasible descent direction $d^1 = d^1(x)$ is then computed. Then for a certain scalar $\rho = \rho(x) \in [0, 1]$, a feasible descent direction $d = (1-\rho)d^0 + \rho d^1$ is obtained, asymptotically close to d^0. Finally a second order correction $\tilde{d} = \tilde{d}(x, d, H)$ is computed, involving auxiliary function evaluations at $x + d$, and an Armijo type search is performed along the arc $x + td + t^2\tilde{d}$. The purpose of $\tilde{d}$ is to allow a full step of one to be taken close to a solution, thus allowing superlinear convergence to take place. Conditions are given in [2] on $d^1(\cdot)$, $\rho(\cdot)$ and $\tilde{d}(\cdot,\cdot)$ that result in a globally convergent, locally superlinear convergent algorithm.

The algorithm in [3] is somewhat more sophisticated. An essential difference is that while feasibility is still required, the requirement of decrease of the max objective value is replaced by the weaker requirement that the max objective value at the new point be lower than its maximum over the last four iterates. The main payoff is that the auxiliary function evaluations can be dispensed with, except possibly at the first few iterations. First a direction $d^1 = d^1(x)$ is computed, which is feasible even at Karush-Kuhn-Tucker points. Then for a certain scalar $\rho^\ell = \rho^\ell(x) \in [0,1]$, a "local" feasible direction $d^\ell = (1-\rho^\ell)d^0 + \rho^\ell d^1$ is obtained, and at $x + d^\ell$ the objective functions are tested and feasibility is checked. If the requirements pointed out above are satisfied, $x + d^\ell$ is accepted as next iterate. This will always be the case close to a solution. Whenever $x + d^\ell$ is not accepted, a "global" feasible *descent* direction $d^g = (1-\rho^g)d^0 + \rho^g d^1$ is obtained with $\rho^g = \rho^g(x) \in [0, \rho^\ell]$. A second order correction $\tilde{d} = \tilde{d}(x, d^g, H)$ is computed the same way as in [2], and a "nonmonotone" search is performed along the arc $x + td^g + t^2\tilde{d}$. Here the purpose of $\tilde{d}$ is to suitably initialize the sequence for the "four iterate" rule. Conditions are given in [3] on $d^1(\cdot)$, $\rho^\ell(\cdot)$, $\rho^g(\cdot)$, and $\tilde{d}(\cdot,\cdot)$ that result in a globally convergent, locally superlinear convergent algorithm. In [4], the algorithm of [3] is refined for the case of unconstrained minimax problems. The major difference over the algorithm of [3] is that there is no need of d^1. As in [3], $\tilde{d}$ is required to initialize superlinear convergence.

The FSQP implementation corresponds to a specific choice of $d^1(\cdot)$, $\rho(\cdot)$, $\tilde{d}(\cdot,\cdot)$, $\rho^\ell(\cdot)$, and $\rho^g(\cdot)$, with some modifications as follows. If the first algorithm is used, d^1 is computed as a function not only of x but also of d^0 (thus of H), as it appears beneficial to keep d^1 relatively close to d^0. In the case of the second algorithm, the construction of d^ℓ is modified so that the function evaluations at different auxiliary points can be avoided during early iteration when $\rho^g \neq \rho^\ell$; the quadratic program that yields $\tilde{d}$ involves only a subset of "active" functions, thus decreasing the number of function evaluations. The details are given below. The analysis in [2], [3] and [4] can be easily extended to these modified algorithms. Also obvious simplifications are introduced concerning linear constraints: the iterates are allowed (for inequality constraints) or are forced (for equality constraints) to stay on the boundary of these constraints and these constraints are not checked in the line search. Finally, FSQP automatically switches to a "phase 1" mode if the initial guess provided by the user is not in the feasible set.

Below we call FSQP-AL the algorithm with the Armijo line search, and FSQP-NL the algorithm with nonmonotone line search. We make use of the notations

$$f_{I^f}(x) = \max_{i \in I^f}\{f_i(x)\}$$

$$f'(x,d,p) = \max_{i\in I^f}\{f_i(x) + \langle\nabla f_i(x), d\rangle\} - f_{I^f}(x) - \sum_{j=1}^{n_e} p_j\langle\nabla h_j(x), d\rangle$$

and, for any subset $I \subset I^f$,

$$\tilde{f}'_I(x+d, x, \tilde{d}, p) = \max_{i\in I}\{f_i(x+d) + \langle\nabla f_i(x), \tilde{d}\rangle\} - f_I(x+d) - \sum_{j=1}^{n_e} p_j\langle\nabla h_j(x), \tilde{d}\rangle.$$

At each iteration k, the quadratic program $QP(x_k, H_k, p_k)$ that yields the SQP direction d_k^0 is defined at x_k for H_k symmetric positive definite by

$$\begin{array}{ll}\displaystyle\min_{d^0\in R^n} & \frac{1}{2}\langle d^0, H_k d^0\rangle + f'(x_k, d^0, p_k) \\ \text{s.t.} & bl \le x_k + d^0 \le bu \\ & g_j(x_k) + \langle\nabla g_j(x_k), d^0\rangle \le 0, \quad j = 1,\ldots,t_i \\ & h_j(x_k) + \langle\nabla h_j(x_k), d^0\rangle \le 0, \quad j = 1,\ldots,n_e \\ & \langle a_j, x_k + d^0\rangle = b_j, \quad j = 1,\ldots,t_e - n_e.\end{array}$$

Let $\zeta_{k,j}$'s with $\sum_{j=1}^{n_f}\zeta_{k,j} = 1$, $\xi_{k,j}$'s, $\lambda_{k,j}$'s, and $\mu_{k,j}$'s denote the multipliers, for the various objective functions, simple bounds (only n possible active bounds at each iteration), inequality, and equality constraints respectively, associated with this quadratic program. Define the set of active objective functions, for any i such that $\zeta_{k,i} > 0$, by

$$I_k^f(d_k) = \{j \in I^f : |f_j(x_k) - f_i(x_k)| \le 0.2\|d_k\|\cdot\|\nabla f_j(x_k) - \nabla f_i(x_k)\|\} \cup \{j \in I^f : \zeta_{k,j} > 0\}$$

and the set of active constraints by

$$I_k^g(d_k) = \{j \in \{1,\ldots,t_i\} : |g_j(x_k)| \le 0.2\|d_k\|\cdot\|\nabla g_j(x_k)\|\} \cup \{j \in \{1,\ldots,t_i\} : \lambda_{k,j} > 0\}.$$

Algorithm FSQP-AL.

Parameters. $\eta = 0.1$, $\nu = 0.01$, $\alpha = 0.1$, $\beta = 0.5$, $\kappa = 2.1$, $\tau_1 = \tau_2 = 2.5$, $\underline{t} = 0.1$, $\epsilon_1 = 1$, $\epsilon_2 = 10$, $\delta = 5$.

Data. $x_0 \in R^n$, $\epsilon > 0$, $\epsilon_e > 0$ and $p_{0,j} = \epsilon_2$ for $j = 1,\ldots,n_e$.

Step 0: Initialization. Set $k = 0$ and $H_0 =$ the identity matrix. Set $nset = 0$. If x_0 is infeasible for some constraint other than a nonlinear equality constraint, substitute a feasible point, obtained as discussed below. For $j = 1,\ldots,n_e$, replace $h_j(x)$ by $-h_j(x)$ whenever $h_j(x_0) > 0$.

Step 1: Computation of a search arc.

i. Compute d_k^0, the solution of the quadratic program $QP(x_k, H_k, p_k)$. If $\|d_k^0\| \le \epsilon$ and $\sum_{j=1}^{n_e}|h_j(x_k)| \le \epsilon_e$, stop. If $n_i + n_e = 0$ and $n_f = 1$, set $d_k = d_k^0$ and $\tilde{d}_k = 0$ and go to *Step 2*. If $n_i + n_e = 0$ and $n_f > 1$, set $d_k = d_k^0$ and go to *Step 1 iv*.

ii. Compute d_k^1 by solving the strictly convex quadratic program

$$\begin{array}{ll} \min\limits_{d^1\in R^n,\gamma\in R} & \frac{\eta}{2}\langle d_k^0 - d^1, d_k^0 - d^1\rangle + \gamma \\ \text{s.t.} & bl \le x_k + d^1 \le bu \\ & f'(x_k, d^1, p_k) \le \gamma \\ & g_j(x_k) + \langle \nabla g_j(x_k), d^1\rangle \le \gamma, \quad j = 1,\ldots,n_i \\ & \langle c_j, x_k + d^1\rangle \le d_j, \quad j = 1,\ldots,t_i - n_i \\ & h_j(x_k) + \langle \nabla h_j(x_k), d^1\rangle \le \gamma, \quad j = 1,\ldots,n_e \\ & \langle a_j, x_k + d^1\rangle = b_j, \quad j = 1,\ldots,t_e - n_e \end{array}$$

iii. Set $d_k = (1-\rho_k)d_k^0 + \rho_k d_k^1$ with $\rho_k = \|d_k^0\|^\kappa/(\|d_k^0\|^\kappa + v_k)$, where $v_k = \max(0.5, \|d_k^1\|^{\tau_1})$.

iv. Compute $\tilde{d}_k$ by solving the strictly convex quadratic program

$$\begin{array}{ll} \min\limits_{\tilde{d}\in R^n} & \frac{1}{2}\langle (d_k + \tilde{d}), H_k(d_k + \tilde{d})\rangle + f'_{I_k^f(d_k)}(x_k, d_k, \tilde{d}, p_k) \\ \text{s.t.} & bl \le x_k + d_k + \tilde{d} \le bu \\ & g_j(x_k + d_k) + \langle \nabla g_j(x_k), \tilde{d}\rangle \le -\min(\nu\|d_k\|, \|d_k\|^{\tau_2}), \ j \in I_k^g(d_k) \cap \{j : j \le n_i\} \\ & \langle c_{j-n_i}, x_k + d_k + \tilde{d}\rangle \le d_{j-n_i}, \quad j \in I_k^g(d_k) \cap \{j : j > n_i\} \\ & h_j(x_k + d_k) + \langle \nabla h_j(x_k), \tilde{d}\rangle \le -\min(\nu\|d_k\|, \|d_k\|^{\tau_2}), \ j = 1,\ldots,n_e \\ & \langle a_j, x_k + d_k + \tilde{d}\rangle = b_j, \quad j = 1,\ldots,t_e - n_e \end{array}$$

where $f'_{I_k^f(d_k)}(x_k, d_k, \tilde{d}, p_k) = f'(x_k, d_k + \tilde{d}, p_k)$ if $n_f = 1$, and $f'_{I_k^f(d_k)}(x_k, d_k, \tilde{d}, p_k) = \tilde{f}'_{I_k^f(d_k)}(x_k + d_k, x_k, \tilde{d}, p_k)$ if $n_f > 1$. If the quadratic program has no solution or if $\|\tilde{d}_k\| > \|d_k\|$, set $\tilde{d}_k = 0$.

Step 2. Arc search. Let $\delta_k = f'(x_k, d_k, p_k)$ if $n_i + n_e \ne 0$ and $\delta_k = -\langle d_k^0, H_k d_k^0\rangle$ otherwise. Compute t_k, the first number t in the sequence $\{1, \beta, \beta^2, \ldots\}$ satisfying

$$\begin{gathered} f_m(x_k + td_k + t^2\tilde{d}_k, p_k) \le f_m(x_k, p_k) + \alpha t\delta_k \\ g_j(x_k + td_k + t^2\tilde{d}_k) \le 0, \quad j = 1,\ldots,n_i \\ \langle c_{j-n_i}, x_k + td_k + t^2\tilde{d}_k\rangle \le d_{j-n_i}, \quad \forall j > n_i \ \&\ j \notin I_k^g(d_k) \\ h_j(x_k + td_k + t^2\tilde{d}_k) \le 0, \quad j = 1,\ldots,n_e. \end{gathered}$$

Specifically, the line search proceeds as follows. First, the linear constraints that were not used in computing $\tilde{d}_k$ are checked until all of them are satisfied, resulting in a stepsize, say, $\bar{t}_k$. Due to the convexity of linear constraints, these constraints will be satisfied for any $t \le \bar{t}_k$. Then, for $t = \bar{t}_k$, nonlinear constraints are checked first and, for both objectives and constraints, those with nonzero multipliers in the QP yielding d_k^0 are evaluated first. For $t < \bar{t}_k$, the function that caused the previous value of t to be rejected is checked first; all functions of the same type ("objective" or "constraint") as the latter will then be checked first.

Step 3. Updates.

- If $nset > 5n$ and $t_k < \underline{t}$, set $H_{k+1} = H_0$ and $nset = 0$. Otherwise, set $nset = nset + 1$ and compute a new approximation H_{k+1} to the Hessian of the Lagrangian using the BFGS formula with Powell's modification [8].

- Set $x_{k+1} = x_k + t_k d_k + t_k^2 \tilde{d}_k$.

- Solve the unconstrained quadratic problem in $\bar{\mu}$

$$\min_{\bar{\mu} \in R^{t_e}} \quad \| \sum_{j=1}^{n_f} \zeta_{k,j} \nabla f_j(x_k) + \xi_k + \sum_{j=1}^{t_i} \lambda_{k,j} \nabla g_j(x_k) + \sum_{j=1}^{t_e} \bar{\mu}_j \nabla h_j(x_k) \|^2,$$

  where the $\zeta_{k,j}$'s, ξ_k and the $\lambda_{k,j}$'s are the multipliers associated with $QP(x_k, H_k, p_k)$ for the objective functions, variable bounds, and inequality constraints respectively.[3] Update p_k as follows: for $j = 1, \ldots, n_e$,

$$p_{k+1,j} = \begin{cases} p_{k,j} & \text{if } p_{k,j} + \bar{\mu}_j \geq \epsilon_1 \\ \max\{\epsilon_1 - \bar{\mu}_j,\ \delta p_{k,j}\} & \text{otherwise.} \end{cases}$$

- Increase k by 1.

- Go back to *Step 1*.

$\square$

Algorithm FSQP-NL.

Parameters. $\eta = 3.0$, $\nu = 0.01$, $\alpha = 0.1$, $\beta = 0.5$, $\theta = 0.2$, $\bar{\rho} = 0.5$, $\gamma = 2.5$, $\underline{C} = 0.01$, $\underline{d} = 5.0$, $\underline{t} = 0.1$, $\epsilon_1 = 0.1$, $\epsilon_2 = 10$, $\delta = 5$.

Data. $x_0 \in R^n$, $\epsilon > 0$, $\epsilon_e > 0$ and $p_{0,j} = \epsilon_2$ for $j = 1, \ldots, n_e$.

Step 0: Initialization. Set $k = 0$, $H_0 =$ the identity matrix, and $C_0 = \underline{C}$. If x_0 is infeasible for constraints other than nonlinear equality constraints, substitute a feasible point, obtained as discussed below. Set $x_{-3} = x_{-2} = x_{-1} = x_0$ and $nset = 0$. For $j = 1, \ldots, n_e$, replace $h_j(x)$ by $-h_j(x)$ whenever $h_j(x_0) > 0$.

Step 1: Computation of a new iterate.

[3] This is a refinement (saving much computation and memory) of the scheme proposed in [1].

i. Compute d_k^0, the solution of quadratic program $QP(x_k, H_k, p_k)$.

If $\|d_k^0\| \leq \epsilon$ and $\sum_{j=1}^{n_e} |h_j(x_k)| \leq \epsilon_e$, stop. If $n_i + n_e = 0$ and $n_f = 1$, set $d_k = d_k^0$ and $\tilde{d}_k = 0$ and go to *Step 1 viii.* If $n_i + n_e = 0$ and $n_f > 1$, set $\rho_k^\ell = \rho_k^g = 0$ and go to *Step 1 v.*

ii. Compute d_k^1 by solving the strictly convex quadratic program

$$\begin{array}{ll} \min\limits_{d^1 \in R^n, \gamma \in R} & \frac{\eta}{2}\|d^1\|^2 + \gamma \\ \text{s.t.} & bl \leq x_k + d^1 \leq bu \\ & g_j(x_k) + \langle \nabla g_j(x_k), d^1 \rangle \leq \gamma, \quad j = 1, \ldots, n_i \\ & \langle c_j, x_k + d^1 \rangle \leq d_j, \quad j = 1, \ldots, t_i - n_i \\ & h_j(x_k) + \langle \nabla h_j(x_k), d^1 \rangle \leq \gamma, \quad j = 1, \ldots, n_e \\ & \langle a_j, x_k + d^1 \rangle = b_j, \quad j = 1, \ldots, t_e - n_e \end{array}$$

iii. Set $v_k = \min\{C_k\|d_k^0\|^2, \|d_k^0\|\}$. Define values $\rho_{k,j}^g$ for $j = 1, \ldots, n_i$ by $\rho_{k,j}^g$ equal to zero if

$$g_j(x_k) + \langle \nabla g_j(x_k), d_k^0 \rangle \leq -v_k$$

or equal to the maximum ρ in $[0, 1]$ such that

$$g_j(x_k) + \langle \nabla g_j(x_k), (1-\rho)d_k^0 + \rho d_k^1 \rangle \geq -v_k$$

otherwise. Similarly, define values $\rho_{k,j}^h$ for $j = 1, \ldots, n_e$. Let

$$\rho_k^\ell = \max\left\{ \max_{j=1,\ldots,n_i} \{\rho_{k,j}^g\}, \max_{j=1,\ldots,n_e} \{\rho_{k,j}^h\} \right\}.$$

iv. Define ρ_k^g as the largest number ρ in $[0, \rho_k^\ell]$ such that

$$f'(x_k, (1-\rho)d_k^0 + \rho d_k^1, p_k) \leq \theta f'(x_k, d_k^0, p_k).$$

If ($k \geq 1$ & $t_{k-1} < 1$) or ($\rho_k^\ell > \bar{\rho}$), set $\rho_k^\ell = \min\{\rho_k^\ell, \rho_k^g\}$.

v. Construct a "local" direction

$$d_k^\ell = (1 - \rho_k^\ell)d_k^0 + \rho_k^\ell d_k^1.$$

Set $M = 3$, $\delta_k = f'(x_k, d_k^0)$ if $n_i + n_e \neq 0$, and $M = 2$, $\delta_k = -\langle d_k^0, H_k d_k^0 \rangle$ otherwise. If

$$f_m(x_k + d_k^\ell, p_k) \leq \max_{\ell=0,\ldots,M} \{f_m(x_{k-\ell}, p_k)\} + \alpha\delta_k$$

$$g_j(x_k + d_k^\ell) \le 0, \quad j = 1, \ldots, n_i$$

and

$$h_j(x_k + d_k^\ell) \le 0, \quad j = 1, \ldots, n_e,$$

set $t_k = 1$, $x_{k+1} = x_k + d_k^\ell$ and go to *Step 2*.

vi. Construct a "global" direction

$$d_k^g = (1 - \rho_k^g) d_k^0 + \rho_k^g d_k^1.$$

vii. Compute $\tilde{d}_k$ by solving the strictly convex quadratic program

$$\begin{array}{ll} \min\limits_{\tilde{d} \in R^n} & \frac{1}{2}\langle (d_k^g + \tilde{d}), H_k(d_k^g + \tilde{d})\rangle + f'_{I_k^f(d_k^g)}(x_k, d_k^g, \tilde{d}, p_k) \\ \text{s.t.} & bl \le x_k + d_k^g + \tilde{d} \le bu \\ & g_j(x_k + d_k^g) + \langle \nabla g_j(x_k), \tilde{d}\rangle \le -\min(\nu \|d_k^g\|, \ \|d_k^g\|^\tau), \quad j \in I_k^g(d_k^g) \cap \{j : j \le n_i\} \\ & \langle c_{j-n_i}, x_k + d_k^g + \tilde{d}\rangle \le d_{j-n_i}, \quad j \in I_k^g(d_k^g) \cap \{j : j > n_i\} \\ & h_j(x_k + d_k^g) + \langle \nabla h_j(x_k), \tilde{d}\rangle \le -\min(\nu \|d_k^g\|, \ \|d_k^g\|^\tau), \quad j = 1, \ldots, n_e \\ & \langle a_j, x_k + d_k^g + \tilde{d}\rangle = b_j, \quad j = 1, \ldots, t_e - n_e \end{array}$$

where $f'_{I_k^f(d_k^g)}(x_k, d_k^g, \tilde{d}, p_k) = f'(x_k, d_k^g + \tilde{d}, p_k)$ if $n_f = 1$, and $f'_{I_k^f(d_k^g)}(x_k, d_k^g, \tilde{d}, p_k) = \tilde{f}'_{I_k^f(d_k^g)}(x_k + d_k^g, x_k, \tilde{d}, p_k)$ if $n_f > 1$. If the quadratic program has no solution or if $\|\tilde{d}_k\| > \|d_k^g\|$, set $\tilde{d}_k = 0$.

viii. Set $M = 3$, $\delta_k = f'(x_k, d_k^g, p_k)$ if $n_i + n_e \ne 0$, and $M = 2$, $\delta_k = -\langle d_k^g, H_k d_k^g\rangle$ otherwise. Compute t_k, the first number t in the sequence $\{1, \beta, \beta^2, \ldots\}$ satisfying

$$f_m(x_k + t d_k^g + t^2 \tilde{d}_k, p_k) \le \max_{\ell=0,\ldots,M}\{f_m(x_{k-\ell}, p_k)\} + \alpha t \delta_k$$

$$g_j(x_k + t d_k^g + t^2 \tilde{d}_k) \le 0, \quad j = 1, \ldots, n_i$$

$$\langle c_{j-n_i}, x_k + t d_k^g + t^2 \tilde{d}_k\rangle \le d_{j-n_i}, \quad j > n_i \ \&\ j \notin I_k^g(d_k^g)$$

$$h_j(x_k + t d_k^g + t^2 \tilde{d}_k) \le 0, \quad j = 1, \ldots, n_e$$

and set $x_{k+1} = x_k + t_k d_k^g + t_k^2 \tilde{d}_k$.

Specifically, the line search proceeds as follows. First, the linear constraints that were not used in computing $\tilde{d}_k$ are checked until all of them are satisfied, resulting in a stepsize, say, $\bar{t}_k$. Due to the convexity of linear constraints, these constraints will be satisfied for any $t \le \bar{t}_k$. Then, for $t = \bar{t}_k$, nonlinear constraints are checked first and, for both objectives and constraints, those with nonzero multipliers in the QP yielding d_k^0 are evaluated first. For $t < \bar{t}_k$, the function that caused the previous value of t to be rejected is checked first; all functions of the same type ("objective" or "constraint") as the latter will then be checked first.

Step 2. Updates.

- If $nset > 5n$ and $t_k < \underline{t}$, set $H_{k+1} = H_0$ and $nset = 0$. Otherwise, set $nset = nset + 1$ and compute a new approximation H_{k+1} to the Hessian of the Lagrangian using the BFGS formula with Powell's modification[8].

- If $\|d_k^0\| > \underline{d}$, set $C_{k+1} = \max\{0.5C_k, \underline{C}\}$. Otherwise, if $g_j(x_k + d_k^\ell) \leq 0, \quad j = 1, \ldots, n_i$, set $C_{k+1} = C_k$. Otherwise, set $C_{k+1} = 10C_k$.

- Solve the unconstrained quadratic problem in $\bar{\mu}$

$$\min_{\bar{\mu} \in R^{t_e}} \quad \| \sum_{j=1}^{n_f} \zeta_{k,j} \nabla f_j(x_k) + \xi_k + \sum_{j=1}^{t_i} \lambda_{k,j} \nabla g_j(x_k) + \sum_{j=1}^{t_e} \mu_j \nabla h_j(x_k) \|^2,$$

  where the $\zeta_{k,j}$'s, ξ_k and the $\lambda_{k,j}$'s are the multipliers associated with $QP(x_k, H_k, p_k)$ for the objective functions, variable bounds, and inequality constraints respectively.[4]

  Update p_k as follows: for $j = 1, \ldots, n_e$,

$$p_{k+1,j} = \begin{cases} p_{k,j} & \text{if } p_{k,j} + \mu_j \geq \epsilon_1 \\ \max\{\epsilon_1 - \bar{\mu}_j, \ \delta p_{k,j}\} & \text{otherwise.} \end{cases}$$

- Increase k by 1.

- Go back to *Step 1*.

$\square$

Remark: The Hessian matrix is reset in both algorithms whenever stepsize is too small and the updating of the matrix goes through n iterations. This is helpful in some situations where the Hessian matrix becomes singular.

If the initial guess x_0 provided by the user is not feasible for some nonlinear inequality constraint or some linear equality constraint, FSQP first solves a strictly convex quadratic program

$$\begin{array}{ll} \min\limits_{v \in R^n} & \langle v, v \rangle \\ \text{s.t.} & bl \leq x_0 + v \leq bu \\ & \langle c_j, x_0 + v \rangle \leq d_j, \quad j = 1, \ldots, t_i - n_i \\ & \langle a_j, x_0 + v \rangle = b_j, \quad j = 1, \ldots, t_e - n_e. \end{array}$$

Then, starting from the point $x = x_0 + v$, it will iterate, using algorithm FSQP-AL, on the problem

---

[4]See footnote to corresponding step in description of FSQP-AL.

$$\begin{array}{ll} \min\limits_{x \in R^n} & \max\limits_{j=1,\ldots,n_i} \{g_j(x)\} \\ \text{s.t.} & bl \leq x \leq bu \\ & \langle c_j, x \rangle \leq d_j, \quad j = 1, \ldots, t_i - n_i \\ & \langle a_j, x \rangle = b_j, \quad j = 1, \ldots, t_e - n_e \end{array}$$

until $\max\limits_{j=1,\ldots,n_i} \{g_j(x)\} \leq 0$ is achieved. The corresponding iterate x will then be feasible for all constraints other than nonlinear equality constraints of the original problem.

3 Specification of Subroutine FSQPD 3.0c

Only a double precision version of FSQP, FSQPD is currently available. The specification of FSQPD is as follows:

```
      subroutine FSQPD(nparam,nf,nineqn,nineq,neqn,neq,mode,iprint,miter,
     *                 inform,bigbnd,eps,epseqn,udelta,bl,bu,x,f,g,
     *                 iw,iwsize,w,nwsize,obj,constr,gradob,gradcn)
      integer nparam,nf,nineqn,nineq,neqn,neq,mode,iprint,miter,inform,
     *        iwsize,nwsize
      integer iw(iwsize)
      double  precision bigbnd,eps,epseqn,udelta
      double  precision bl(nparam),bu(nparam),x(nparam),
     *        f(nf),g(nineq+neq),w(nwsize)
      external obj,constr,gradob,gradcn
```

Important: all real variables and arrays must be declared as double precision in the routine that calls FSQPD. The following are specifications of parameters and workspace.

`nparam` **(Input)** Number of free variables, i.e., the dimension of `x`.

`nf` **(Input)** Number of objective functions (n_f in the algorithm description).

`nineqn` **(Input)** Number (possibly zero) of nonlinear inequality constraints (n_i in the algorithm description).

`nineq` **(Input)** Total number (possibly equal to `nineqn`) of inequality constraints (t_i in the algorithm description).

`neqn` **(Input)** Number (possibly zero) of nonlinear equality constraints (n_e in the algorithm description).

`neq` **(Input)** Total Number (possibly equal to `neqn`) of equality constraints (t_e in the algorithm description).

`mode` **(Input)** `mode` $= 1BA$ with the following meanings:

- `A` $= 0$: (P) is to be solved.
- `A` $= 1$: (PL_∞) is to be solved. (PL_∞) is defined as follows

$$(PL_\infty) \qquad \min \max_{i \in I^f} |f_i(x)| \quad \text{s.t.} \quad x \in X$$

  where X is the same as for (P). It is handled in this code by splitting $|f_i(x)|$ as $f_i(x)$ and $-f_i(x)$ for each i. The user is required to provide only $f_i(x)$ for $i \in I^f$.
- `B` $= 0$: Algorithm FSQP-AL is selected, resulting in a decrease of the (modified) objective function at each iteration.
- `B` $= 1$: Algorithm FSQP-NL is selected, resulting in a decrease of the (modified) objective function within at most four iterations (or three iterations, see Algorithm FSQP-NL).

`iprint` **(Input)** Parameter indicating the desired output (see §4 for details):

- `iprint` $= 0$: No information except for user-input errors is displayed. This value is imposed during phase 1.
- `iprint` $= 1$: At the end of execution, status (`inform`), iterate, objective values, constraint values, number of evaluations of objectives and nonlinear constraints, norm of the Kuhn-Tucker vector, and sum of feasibility violation are displayed.
- `iprint` $= 2$: At the end of each iteration, the same information as with `iprint` $= 1$ is displayed.
- `iprint` $= 3$: At each iteration, the same information as with `iprint` $= 2$, including detailed information on the search direction computation, on the line search, and on the update is displayed.

`miter` **(Input)** Maximum number of iterations allowed by the user before termination of execution.

`inform` **(Output)** Parameter indicating the status of the execution of FSQPD:

- $\texttt{inform} = 0$: Normal termination of execution in the sense that $\|d^0\| \leq$ `eps` and (if `neqn` $\neq 0$) $\sum_{j=1}^{n_e} |h_j(x)| \leq$ `epseqn`.
- $\texttt{inform} = 1$: The user-provided initial guess is infeasible for linear constraints and FSQPD is unable to generate a point satisfying all these constraints.
- $\texttt{inform} = 2$: The user-provided initial guess is infeasible for nonlinear inequality constraints and linear constraints; and FSQPD is unable to generate a point satisfying all these constraints.
- $\texttt{inform} = 3$: The maximum number `miter` of iterations has been reached before a solution is obtained.
- $\texttt{inform} = 4$: The line search fails to find a new iterate (trial step size being smaller than the machine precision `epsmac` computed by FSQPD).
- $\texttt{inform} = 5$: Failure in attempting to construct d^0.
- $\texttt{inform} = 6$: Failure in attempting to construct d^1.
- $\texttt{inform} = 7$: Input data are not consistent (with printout indicating the error).

`bigbnd` **(Input)** (see also `bl` and `bu` below) It plays the role of Infinite Bound.

`eps` **(Input)** Final norm requirement for the Newton direction d_k^0 (ϵ in the algorithm description). It must be bigger than the machine precision `epsmac` (computed by FSQPD). (If the user does not have a good feeling of what value should be chosen, a very small number could be provided and `iprint` $= 2$ be selected so that the user would be able to keep trace of the process of optimization and terminate FSQPD at appropriate time.)

`epseqn` **(Input)** Maximum violation of nonlinear equality constraints allowed by the user at an optimal point (ϵ_e in the algorithm description). It is in effect only if $n_e \neq 0$ and must be bigger than the machine precision `epsmac` (computed by FSQPD).

`udelta` **(Input)** The perturbation size the user suggests to use in approximating gradients by finite difference. The perturbation size actually used is defined by $\text{sign}(x^i) \times \max\{\texttt{udelta}, \ \texttt{rteps} \times \max(1, |x^i|)\}$ for each component x^i of x

(`rteps` is the square root of `epsmac`). `udelta` should be set to zero if the user has no idea how to choose it.

`bl` **(Input)** Array of dimension `nparam` containing lower bounds for the components of `x`. To specify a non-existent lower bound (i.e., $\texttt{bl}(j) = -\infty$ for some j), the value used must satisfy $\texttt{bl}(j) \leq -\texttt{bigbnd}$.

`bu` **(Input)** Array of dimension `nparam` containing upper bounds for the components of `x`. To specify a non-existent upper bound (i.e., $\texttt{bu}(j) = \infty$ for some j), the value used must satisfy $\texttt{bu}(j) \geq \texttt{bigbnd}$.

`x` **(Input)** Initial guess.
(Output) Iterate at the end of execution.

`f` Array of dimension $\max\{1, \texttt{nf}\}$.
(Output) Value of functions $f_i, i = 1, \ldots, n_f$, at `x` at the end of execution.

`g` Array of dimension $\max\{1, \texttt{nineq} + \texttt{neq}\}$.
(Output) Values of constraints at `x` at the end of execution.

`iw` Workspace vector of dimension `iwsize`.

`iwsize` **(Input)** Workspace length for `iw`. It must be at least as big as $6 \times \texttt{nparam} + 8 \times (\texttt{nineq} + \texttt{neq}) + 7 \times \texttt{nf} + 30$. This estimate is usually very conservative and the smallest suitable value will be displayed if the user-supplied value is too small.

`w` **(Input)** Workspace of dimension `nwsize`.
(Output) Lagrange multipliers in the first $\texttt{nparam} + \texttt{nineq} + \texttt{neq} + \texttt{nff}$ entries; where $\texttt{nff} = 0$ if (in `mode`) $\texttt{A} = 0$ and $\texttt{nf} = 1$, and $\texttt{nff} = \texttt{nf}$ otherwise. They are ordered as ξ's (variables), λ's (inequality constraints), μ's (equality constraints), and ζ (objective functions). $\lambda_j \geq 0 \;\; \forall j = 1, \ldots, t_i$ and $\mu_j \geq 0 \;\; \forall j = 1, \ldots, t_e$. $\xi_i > 0$ indicates that x_i reaches its upper bound and $\xi_i < 0$ indicates that x_i reaches its lower bound. When (in `mode`) $\texttt{A} = 0$ and $\texttt{nf} > 1$, $\zeta_i \geq 0$. When $\texttt{B} = 1$, $\zeta_i > 0$ refers to $+f_i(x)$ and $\zeta_i < 0$ to $-f_i(x)$.

`nwsize` **(Input)** Workspace length for `w`. It must be at least as big as $4 \times \texttt{nparam}^2 + 5 \times (\texttt{nineq} + \texttt{neq}) \times \texttt{nparam} + 3 \times \texttt{nf} \times \texttt{nparam} + +26 \times (\texttt{nparam} + \texttt{nf}) + 45 \times (\texttt{nineq} + \texttt{neq}) + 100$. This estimate is usually very conservative and the smallest suitable value will be displayed if the user-supplied value is too small.

`obj` **(Input)** Name of the user-defined subroutine that computes the value of the objective functions $f_i(x)$, $\forall i = 1, \ldots, n_f$. This name must be declared as **external** in the calling routine and passed as an argument to FSQPD. The detailed specification is given in §5.1 below.

`constr` **(Input)** Name of the user-defined subroutine that computes the value of the constraints. This name must be declared as **external** in the calling routine and passed as an argument to FSQPD. The detailed specification is given in §5.2 below.

`gradob` **(Input)** Name of the subroutine that computes the gradients of the objective functions $f_i(x)$, $\forall i = 1, \ldots, n_f$. This name must be declared as **external** in the calling routine and passed as an argument to FSQPD. The user must pass the subroutine name `grobfd` (and declare it as **external**), if he/she wishes that FSQPD evaluate these gradients automatically, by forward finite differences. The detailed specification is given in §5.3 below.

`gradcn` **(Input)** Name of the subroutine that computes the gradients of the constraints. This name must be declared as **external** in the calling routine and passed as an argument to FSQPD. The user must pass the subroutine name `grcnfd` (and declare it as **external**), if he/she wishes that FSQPD evaluate these gradients automatically, by forward finite differences. The detailed specification is given in §5.4 below.

4 User-Accessible Stopping Criterion

As is clear from the two algorithms, the optimization process normally terminates if both $\|d_k^0\| \leq \epsilon$ and $\sum_{j=1}^{n_e} |h_j(x_k)| \leq \epsilon_e$ are satisfied. Very small value of either of these two parameters may request exceedingly long execution time, depending on the complexity of underlying problem and the nonlinearity of various functions. FSQP allows users to specify their own stopping criterion in any one of the four user-supplied subroutines mentioned above via the following common block

```
integer nstop
common /fsqpst/nstop
```

if (s)he wishes to. `nstop` = 0 should be returned to FSQP when the stopping criterion is satisfied. FSQP will check the value of `nstop` at appropriate places during the optimization process and will terminate when either the user's criterion or the default criterion is satisfied.

5 Description of the Output

No output will be displayed before a feasible starting point is obtained. The following information is displayed at the end of execution when `iprint` $= 1$ or at each iteration when `iprint` $= 2$:

- **`iteration`** Total number of iterations (`iprint` $= 1$) or iteration number (`iprint` $= 2$).
- **`inform`** See §3. It is displayed only at the end of execution.
- **`x`** Iterate.
- **`objectives`** Value of objective functions $f_i(x)$, $\forall i = 1, \ldots, n_f$ at `x`.
- **`objmax`** (displayed only if `nf` > 1) The maximum value of the set of objective functions (i.e., $\max f_i(x)$ or $\max |f_i(x)|$, $\forall i = 1, \ldots, n_f$) at `x`.
- **`objective max4`** (displayed only if `B` $= 1$ in `mode`) Largest value of the maximum of the objective functions over the last four (or three, see FSQP-NL) iterations (including the current one).
- **`constraints`** Values of the constraints at `x`.
- **`ncallf`** Number of evaluations (so far) of individual (scalar) objective function $f_i(x)$ for $1 \le i \le n_f$.
- **`ncallg`** Number of evaluations (so far) of individual (scalar) nonlinear constraints.
- **`d0norm`** Norm of the Newton direction d_k^0.
- **`ktnorm`** Norm of the Kuhn-Tucker vector at the current iteration. The Kuhn-Tucker vector is given by

$$\begin{aligned}\nabla L(x_k, \zeta_k, \xi_k, \lambda_k, \mu_k, p_k) &= \sum_{j=1}^{n_f} \zeta_{k,j} \nabla f_j(x_k) + \xi_k + \sum_{j=1}^{t_i} \lambda_{k,j} \nabla g_j(x_k) \\ &\quad + \sum_{j=1}^{n_e} (\mu_{k,j} - p_{k,j}) \nabla h_j(x_k) + \sum_{j=n_e+1}^{t_e} \mu_{k,j} \nabla h_j(x_k).\end{aligned}$$

- **`SCV`** Sum of the violation of nonlinear equality constraints at a solution.

For `iprint` $= 3$, in addition to the same information as the one for `iprint` $= 2$, the following is printed at every iteration.

Details in the computation of a search direction:

`d0` Quasi-Newton direction d_k^0.

`d1` First order direction d_k^1.

`d1norm` Norm of d_k^1.

`d` (`B` $= 0$ in `mode`) Feasible descent direction $d_k = (1 - \rho_k)d_k^0 + \rho_k d_k^1$.

`dnorm` (`B` $= 0$ in `mode`) Norm of d_k.

`rho` (`B` $= 0$ in `mode`) Coefficient ρ_k in constructing d_k.

`dl` (`B` $= 1$ in `mode`) Local direction $d_k^\ell = (1 - \rho_k^\ell)d_k^0 + \rho_k^\ell d_k^1$.

`dlnorm` (`B` $= 1$ in `mode`) Norm of d_k^ℓ.

`rhol` (`B` $= 1$ in `mode`) Coefficient ρ_k^ℓ in constructing d_k^ℓ.

`dg` (`B` $= 1$ in `mode`) Global search direction $d^g = (1 - \rho_k^g)d_k^0 + \rho_k^g d_k^1$.

`dgnorm` (`B` $= 1$ in `mode`) Norm of d_k^g.

`rhog` (`B` $= 1$ in `mode`) Coefficient ρ_k^g in constructing d_k^g.

`dtilde` Second order correction $\tilde{d}_k$.

`dtnorm` Norm of $\tilde{d}_k$.

Details in the line search:

`trial step` Trial steplength t in the search direction.

`trial point` Trial iterate along the search arc with `trial step`.

`trial objectives` This gives the indices i and the corresponding values of the functions $f_i(x) - \sum_{j=1}^{n_e} p_j h_j(x)$ for $1 \leq i \leq n_f$ up to the one which fails in line search at the `trial point`. The indices i are not necessarily in the natural order (see remark at the end of *Step 2* in FSQP-AL and of the end of *Step 1 viii* in FSQP-NL).

`trial constraints` This gives the indices j and the corresponding values of nonlinear constraints for $1 \leq j \leq n_i + n_e$ up to the one which is not feasible at the `trial point`. The indices j are not necessarily in the natural order (see remark at the end of *Step 2* in FSQP-AL and of the end of *Step 1 viii* in FSQP-NL).

Details in the updates:

delta Perturbation size for each variable in finite difference gradients computation.

gradf Gradients of functions $f_i(x)$, $\forall i = 1, \ldots, n_f$, at the new iterate.

gradg Gradients of constraints at the new iterate.

p Penalty parameters for nonlinear equality constraints at the new iterate.

multipliers Multiplier estimates ordered as ξ's, λ's, μ's, and ζ's (from quadratic program computing d_k^0). $\lambda_j \geq 0 \quad \forall j = 1, \ldots, t_i$ and $\mu_j \geq 0 \quad \forall j = 1, \ldots, t_e$. $\xi_i > 0$ indicates that x_i reaches its upper bound and $\xi_i < 0$ indicates that x_i reaches its lower bound. When (in `mode`) `A` $= 0$ and `nf` > 1, $\zeta_i \geq 0$. When (in `mode`) `A` $= 1$, $\zeta_i > 0$ refers to $+f_i(x)$ and $\zeta_i < 0$ to $-f_i(x)$. (cf. §3 under item `w`.)

hess New estimate of the Hessian matrix of the Lagrangian.

Ck The value C_k as defined in Algorithm FSQP-NL.

6 User-Supplied Subroutines

At least two of the following four Fortran 77 subroutines, namely `obj` and `constr`, must be provided by the user in order to define the problem. The name of all four routines can be changed at the user's will, as they are passed as arguments to FSQPD.

6.1 Subroutine obj

The subroutine **obj**, to be provided by the user, computes the value of the objective functions. A (dummy) subroutine must be provided due to Fortran 77 compiling requirement if `nf` $=$ 0 (This may happen when the user is only interested in finding a feasible point). The specification of **obj** for FSQPD is

```
      subroutine obj(nparam,j,x,fj)
      integer nparam,j
      double precision x(nparam),fj
c
c     for given j, assign to fj the value of the jth objective
c     evaluated at x
c
      return
      end
```

Arguments:

`nparam` **(Input)** Dimension of `x`.

`j` **(Input)** Number of the objective to be computed.

`x` **(Input)** Current iterate.

`fj` **(Output)** Value of the `j`th objective function at `x`.

6.2 Subroutine constr

The subroutine **constr**, to be provided by the user, computes the value of the constraints. If there are no constraints, a (dummy) subroutine must be provided anyway due to Fortran 77 compiling requirement. The specification of `constr` for FSQPD is as follows

```
      subroutine constr(nparam,j,x,gj)
      integer nparam,j
      double precision x(nparam),gj
c
c     for given j, assign to gj the value of the jth constraint
c     evaluated at x
c
      return
      end
```

Arguments:

`nparam` **(Input)** Dimension of `x`.

`j` **(Input)** Number of the constraint to be computed.

`x` **(Input)** Current iterate.

`gj` **(Output)** Value of the `j`th constraint at `x`.

The order of the constraints must be as follows. First the `nineqn` (possibly zero) nonlinear inequality constraints. Then the `nineq` − `nineqn` (possibly zero) linear inequality constraints. Finally, the **`neqn`** (possibly zero) nonlinear equality constraints followed by the `neq` − `neqn` (possibly zero) linear equality constraints.

6.3 Subroutine gradob

The subroutine **gradob** computes the gradients of the objective functions. The user may omit to provide this routine and require that forward finite difference approximation be used by FSQPD via calling `grobfd` instead (see argument `gradob` of FSQPD in §3). The specification of `gradob` for FSQPD is as follows

```
      subroutine gradob(nparam,j,x,gradfj,dummy)
      integer nparam,j
      double precision x(nparam),gradfj(nparam)
      double precision dummy
      external dummy
c
c     assign to gradfj the gradient of the jth objective function
c     evaluated at x
c
      return
      end
```

Arguments:

`nparam` **(Input)** Dimension of `x`.

`j` **(Input)** Number of objective for which gradient is to be computed.

`x` **(Input)** Current iterate.

`gradfj` **(Output)** Gradient of the `j`th objective function at x.

`dummy` **(Input)** Used by `grobfd`.

Note that `dummy` is passed as arguments to `gradob` to allow for forward finite difference computation of the gradient.

6.4 Subroutine gradcn

The subroutine **gradcn** computes the gradients of the constraints. The user may omit to provide this routine and require that forward finite difference approximation be used by FSQPD via calling `grcnfd` instead (see argument `gradcn` of FSQPD in §3). The specification of `gradcn` for FSQPD is as follows

```
      subroutine gradcn (nparam,j,x,gradgj,dummy)
      integer nparam,j
      double precision x(nparam),gradgj(nparam)
      double precision dummy
      external dummy
c
c     assign to gradgj the gradient of the jth constraint
c     evaluated at x
c
      return
      end
```

Arguments:

`nparam`	**(Input)** Dimension of `x`.
`j`	**(Input)** Number of constraint for which gradient is to be computed.
`x`	**(Input)** Current iterate.
`gradgj`	**(Output)** Gradient of the `j`th constraint evaluated at `x`.
`dummy`	**(Input)** Used by `grcnfd`.

Note that `dummy` is passed as arguments to `gradcn` to allow for forward finite difference computation of the gradients.

7 Organization of FSQPD and Main Subroutines

7.1 Main Subroutines

FSQPD first checks for inconsistencies of input parameters using the subroutine `check`. It then checks if the starting point given by the user satisfies the linear constraints and if not, generates a point satisfying these constraints using subroutine `initpt`. It then calls FSQPD1 for generating a point satisfying linear and nonlinear inequality constraints. Finally, it attempts to find a point satisfying the optimality condition using again FSQPD1.

`check` Check that all upper bounds on variables are no smaller than lower bounds; check that all input integers are nonnegative and appropriate (`nineq` $\geq$ `nineqn`, etc.); and check that `eps` (ϵ) and (if `neqn` $\neq 0$) `epseqn` (ϵ_e) are at least as large as the machine precision `epsmac` (computed by FSQPD).

`initpt` Attempt to generate a feasible point satisfying simple bounds and all linear constraints.

`FSQPD1` Main subroutine used possibly twice by FSQPD, first for generating a feasible iterate as explained at the end of §2 and second for generating an optimal iterate from that feasible iterate.

FSQPD1 uses the following subroutines:

`dir` Compute various directions d_k^0, d_0^1 and $\tilde{d}_k$.

`step` Compute a step size along a certain search direction. It is also called to check if $x_k + d_k^\ell$ is acceptable in *Step 1 v* of Algorithm FSQP-NL.

`hesian` Perform the Hessian matrix updating.

`out` Print the output for `iprint` = 1 or `iprint` = 2.

`grobfd` (optional) Compute the gradient of an objective function by forward finite differences with mesh size equal to $\text{sign}(x^i) \times \max\{$`udelta`, `rteps` $\times \max(1, |x^i|)\}$ for each component x^i of x (`rteps` is the square root of `epsmac`, the machine precision computed by FSQPD).

`grcnfd` (optional) Compute the gradient of a constraint by forward finite differences with mesh size equal to $\text{sign}(x^i) \times \max\{$`udelta`, `rteps` $\times \max(1, |x^i|)\}$ for each component x^i of x (`rteps` is the square root of `epsmac`, the machine precision computed by FSQPD).

7.2 Other Subroutines

In addition to QLD, the following subroutines are used:

```
diagnl  di1     dqp     error   estlam  fool    fuscmp  indexs  matrcp
matrvc  nullvc  resign  sbout1  sbout2  scaprd  shift   slope   small
```

7.3 Reserved Common Blocks

The following named common blocks are used in FSQPD and QLD:

```
fsqpp1  fsqpp2  fsqpp3  fsqpq1  fsqpq2  fsqplo  fsqpqp  fsqpst  CMACHE
```

8 Examples

The first problem is borrowed from [9] (Problem 32). It involves a single objective function, simple bounds on the variables, nonlinear inequality constraints, and linear equality constraints. The objective function f is defined for $x \in R^3$ by

$$f(x) = (x_1 + 3x_2 + x_3)^2 + 4(x_1 - x_2)^2$$

The constraints are

$$\begin{array}{ll} 0 \leq x_i, & i = 1, \ldots, 3 \\ x_1^3 - 6x_2 - 4x_3 + 3 \leq 0 & 1 - x_1 - x_2 - x_3 = 0 \end{array}$$

The feasible initial guess is: $x_0 = (0.1,\ 0.7,\ 0.2)^T$ with corresponding value of the objective function $f(x_0) = 7.2$. The final solution is: $x^* = (0,\ 0,\ 1)^T$ with $f(x^*) = 1$. A suitable main program is as follows.

```
c
c     problem description
c
      program sampl1
c
      integer iwsize,nwsize,nparam,nf,nineq,neq
      parameter (iwsize=29, nwsize=219)
      parameter (nparam=3, nf=1)
      parameter (nineq=1, neq=1)
      integer iw(iwsize)
      double  precision x(nparam),bl(nparam),bu(nparam),
     *        f(nf+1),g(nineq+neq+1),w(nwsize)
      external obj32,cntr32,grob32,grcn32
c
      integer mode,iprint,miter,nineqn,neqn,inform
      double precision bigbnd,eps,epseqn,udelta
c
      mode=100
      iprint=1
      miter=500
      bigbnd=1.d+10
      eps=1.d-08
      epseqn=0.d0
      udelta=0.d0
```

```
c
c     nparam=3
c     nf=1
      nineqn=1
      neqn=0
c     nineq=1
c     neq=1
c
      bl(1)=0.d0
      bl(2)=0.d0
      bl(3)=0.d0
      bu(1)=bigbnd
      bu(2)=bigbnd
      bu(3)=bigbnd
c
c     give the initial value of x
c
      x(1)=0.1d0
      x(2)=0.7d0
      x(3)=0.2d0
c
      call FSQPD(nparam,nf,nineqn,nineq,neqn,neq,mode,iprint,
     *           miter,inform,bigbnd,eps,epseqn,udelta,bl,bu,x,f,g,
     *           iw,iwsize,w,nwsize,obj32,cntr32,grob32,grcn32)
      end
```

Following are the subroutines defining the objective and constraints and their gradients.

```
      subroutine obj32(nparam,j,x,fj)
      integer nparam,j
      double precision x(nparam),fj
c
      fj=(x(1)+3.d0*x(2)+x(3))**2+4.d0*(x(1)-x(2))**2
      return
      end
c
      subroutine grob32(nparam,j,x,gradfj,dummy)
      integer nparam,j
      double  precision x(nparam),gradfj(nparam),dummy,fa,fb
```

```
      external dummy
c
      fa=2.d0*(x(1)+3.d0*x(2)+x(3))
      fb=8.d0*(x(1)-x(2))
      gradfj(1)=fa+fb
      gradfj(2)=fa*3.d0-fb
      gradfj(3)=fa
      return
      end
c
      subroutine cntr32(nparam,j,x,gj)
      integer nparam,j
      double precision x(nparam),gj
      external dummy
c
      go to (10,20),j
 10   gj=x(1)**3-6.0d0*x(2)-4.0d0*x(3)+3.d0
      return
 20   gj=1.0d0-x(1)-x(2)-x(3)
      return
      end
c
      subroutine grcn32(nparam,j,x,gradgj,dummy)
      integer nparam,j
      double  precision x(nparam),gradgj(nparam),dummy
c
      go to (10,20),j
 10   gradgj(1)=3.d0*x(1)**2
      gradgj(2)=-6.d0
      gradgj(3)=-4.d0
      return
 20   gradgj(1)=-1.d0
      gradgj(2)=-1.d0
      gradgj(3)=-1.d0
      return
      end
```

The file containing the user-provided subroutines is then compiled together with `fsqpd.f` and `qld.f`. After running the algorithm on a SUN 4/SPARC station 1, the following output is obtained:

```
FSQP Version 3.0c (Released September 1992)
Copyright (c) 1989 --- 1992
J.L. Zhou and A.L. Tits
All Rights Reserved


The given initial point is feasible for inequality
       constraints and linear equality constraints:
                     0.10000000000000E+00
                     0.70000000000000E+00
                     0.20000000000000E+00

iteration                               3
inform                                  0
x                   -0.98607613152626E-31
                     0.00000000000000E+00
                     0.10000000000000E+01
objectives           0.10000000000000E+01
constraints         -0.10000000000000E+01
                     0.00000000000000E+00
SCV                  0.00000000000000E+00
d0norm               0.13945222387368E-30
ktnorm               0.10609826585190E-29
ncallf                                  3
ncallg                                  5


Normal termination: You have obtained a solution !!
```

Our second example is taken from example 6 in [10]. The problem is as follows.

$$
\begin{array}{lll}
\displaystyle\min_{x\in R^6} & \displaystyle\max_{i=1,\dots,163} |f_i(x)| & \\
\text{s.t.} & -x(1) & +s \le 0 \\
& x(1)-x(2) & +s \le 0 \\
& x(2)-x(3) & +s \le 0 \\
& x(3)-x(4) & +s \le 0 \\
& x(4)-x(5) & +s \le 0 \\
& x(5)-x(6) & +s \le 0 \\
& x(6)-3.5 & +s \le 0;
\end{array}
$$

where

$$
\begin{array}{l}
f_i(x) = \frac{1}{15} + \frac{2}{15}(\sum_{j=1}^{6} \cos(2\pi x_j \sin\theta_i) + \cos(7\pi \sin\theta_i)), \\
\theta_i = \frac{\pi}{180}(8.5 + 0.5i), \; i = 1, \dots, 163, \\
s = 0.425.
\end{array}
$$

The feasible initial guess is: $x_0 = (0.5, 1, 1.5, 2, 2.5, 3)^T$ with the corresponding value of the objective function $\displaystyle\max_{i=1,\dots,163} |f_i(x_0)| = 0.22051991555531$. A suitable main program is as follows.

```
c
c     problem description
c
      program sampl2
c
      integer iwsize,nwsize,nparam,nf,nineq,neq
      parameter (iwsize=1029, nwsize=7693)
      parameter (nparam=6, nf=163)
      parameter (nineq=7, neq=0)
      integer iw(iwsize)
      double  precision x(nparam),bl(nparam),bu(nparam),
     *        f(nf+1),g(nineq+neq+1),w(nwsize)
      external objmad,cnmad,grobfd,grcnfd
c
      integer mode,iprint,miter,nineqn,neqn,inform
      double precision bigbnd,eps,udelta
c
      mode=111
      iprint=1
      miter=500
```

```
      bigbnd=1.d+10
      eps=1.0d-08
      epseqn=0.d0
      udelta=0.d0
c
c     nparam=6
c     nf=163
      nineqn=0
      neqn=0
c     nineq=7
c     neq=0
c
      bl(1)=-bigbnd
      bl(2)=-bigbnd
      bl(3)=-bigbnd
      bl(4)=-bigbnd
      bl(5)=-bigbnd
      bl(6)=-bigbnd
      bu(1)=bigbnd
      bu(2)=bigbnd
      bu(3)=bigbnd
      bu(4)=bigbnd
      bu(5)=bigbnd
      bu(6)=bigbnd
c
c     give the initial value of x
c
      x(1)=0.5d0
      x(2)=1.d0
      x(3)=1.5d0
      x(4)=2.d0
      x(5)=2.5d0
      x(6)=3.d0
c
      call FSQPD(nparam,nf,nineqn,nineq,neqn,neq,mode,iprint,
     *           miter,inform,bigbnd,eps,epseqn,udelta,bl,bu,x,f,g,
     *           iw,iwsize,w,nwsize,objmad,cnmad,grobfd,grcnfd)
      end
```

```
      stop
```

We choose to compute the gradients of functions by means of finite difference approximation. Thus only subroutines that define the objectives and constraints are needed as follows.

```
      subroutine objmad(nparam,j,x,fj)
      integer nparam,j,i
      double precision x(nparam),theta,pi,fj
c
      pi=3.14159265358979d0
      theta=pi*(8.5d0+dble(j)*0.5d0)/180.d0
      fj=0.d0
      do 10 i=1,6
 10     fj=fj+dcos(2.d0*pi*x(i)*dsin(theta))
      fj=2.d0*(fj+dcos(2.d0*pi*3.5d0*dsin(theta)))/15.d0
     *  +1.d0/15.d0
      return
      end
c
      subroutine cnmad(nparam,j,x,gj)
      integer nparam,j
      double precision x(nparam),ss,gj
c
      ss=0.425d0
      goto(10,20,30,40,50,60,70),j
 10   gj=ss-x(1)
      return
 20   gj=ss+x(1)-x(2)
      return
 30   gj=ss+x(2)-x(3)
      return
 40   gj=ss+x(3)-x(4)
      return
 50   gj=ss+x(4)-x(5)
      return
 60   gj=ss+x(5)-x(6)
      return
 70   gj=ss+x(6)-3.5d0
      return
      end
```

After running the algorithm on a SUN 4/SPARC station 1, the following output is obtained (the results for the set of objectives have been deleted to save space)

```
                FSQP Version 3.0c (Released September 1992)
                      Copyright (c) 1989 --- 1992
                        J.L. Zhou and A.L. Tits
                          All Rights Reserved


The given initial point is feasible for inequality
       constraints and linear equality constraints:
                          0.50000000000000E+00
                          0.10000000000000E+01
                          0.15000000000000E+01
                          0.20000000000000E+01
                          0.25000000000000E+01
                          0.30000000000000E+01

iteration                                    7
inform                                       0
x                         0.42500000000000E+00
                          0.85000000000000E+00
                          0.12750000000000E+01
                          0.17000000000000E+01
                          0.21840763196688E+01
                          0.28732755096448E+01
objective max4            0.11421841325221E+00
objmax                    0.11310472749826E+00
constraints               0.00000000000000E+00
                          0.00000000000000E+00
                          0.00000000000000E+00
                          0.00000000000000E+00
                         -0.59076319668817E-01
                         -0.26419918997596E+00
                         -0.20172449035522E+00
SCV                       0.00000000000000E+00
d0norm                    0.15662162275640E-09
ktnorm                    0.20564110435030E-10
```

```
ncallf                              1141


Normal termination: You have obtained a solution !!
```

Our third example is borrowed from [9] (Problem 71). It involves both equality and inequality nonlinear constraints and is defined by

$$\begin{array}{ll}\min\limits_{x\in R^4} & x_1x_4(x_1+x_2+x_3)+x_3 \\ \text{s.t.} & 1\le x_i\le 5, \quad i=1,\dots,4 \\ & x_1x_2x_3x_4-25\ge 0 \\ & x_1^2+x_2^2+x_3^2+x_4^2-40=0.\end{array}$$

The feasible initial guess is: $x_0=(1,5,5,1)^T$ with the corresponding value of the objective function $f(x_0)=16$. A suitable program that invokes FSQP to solve this problem is given below.

```
c
c     problem description
c
      integer iwsize,nwsize,nparam,nf,nineq,neq
      parameter (iwsize=33, nwsize=284)
      parameter (nparam=4, nf=1)
      parameter (nineq=1, neq=1)
      integer iw(iwsize)
      double  precision x(nparam),bl(nparam),bu(nparam),f(nf+1),
     *        g(nineq+neq+1),w(nwsize)
      external obj,cntr,gradob,gradcn
c
      integer mode,iprint,miter,neqn,nineqn,inform
      double precision bigbnd,eps,epseqn,udelta
c
      mode=100
      iprint=1
      miter=500
      bigbnd=1.d+10
      eps=1.0d-07
      epseqn=7.d-06
      udelta=0.d0
c
```

```
      neqn=1
      nineqn=1
c
      bl(1)=1.d0
      bl(2)=1.d0
      bl(3)=1.d0
      bl(4)=1.d0
      bu(1)=5.d0
      bu(2)=5.d0
      bu(3)=5.d0
      bu(4)=5.d0
c
c     give the initial value of x
c
      x(1)=1.d0
      x(2)=5.d0
      x(3)=5.d0
      x(4)=1.d0
c
      call FSQPD(nparam,nf,nineqn,nineq,neqn,neq,mode,iprint,
     *           miter,inform,bigbnd,eps,epseqn,udelta,bl,bu,x,f,g,
     *           iw,iwsize,w,nwsize,obj,cntr,gradob,gradcn)
      end
```

Following are the subroutines that define the objective, constraints and their gradients.

```
      subroutine obj(nparam,j,x,fj)
      integer nparam,j
      double precision x(nparam),fj
c
      fj=x(1)*x(4)*(x(1)+x(2)+x(3))+x(3)
      return
      end
c
      subroutine gradob(nparam,j,x,gradfj,dummy)
      integer nparam,j
      double precision dummy,x(nparam),gradfj(nparam)
      external dummy
c
```

```
      gradfj(1)=x(4)*(x(1)+x(2)+x(3))+x(1)*x(4)
      gradfj(2)=x(1)*x(4)
      gradfj(3)=x(1)*x(4)+1.d0
      gradfj(4)=x(1)*(x(1)+x(2)+x(3))
      return
      end
c
      subroutine cntr(nparam,j,x,gj)
      integer nparam,j
      double precision x(nparam),gj
c
      goto (10,20),j
 10   gj=25.d0-x(1)*x(2)*x(3)*x(4)
      return
 20   gj=x(1)**2+x(2)**2+x(3)**2+x(4)**2-40.d0
      return
      end
c
      subroutine gradcn(nparam,j,x,gradgj,dummy)
      integer nparam,j
      double precision dummy,x(nparam),gradgj(nparam)
      external dummy
c
      goto (10,20),j
 10   gradgj(1)=-x(2)*x(3)*x(4)
      gradgj(2)=-x(1)*x(3)*x(4)
      gradgj(3)=-x(1)*x(2)*x(4)
      gradgj(4)=-x(1)*x(2)*x(3)
      return
 20   gradgj(1)=2.d0*x(1)
      gradgj(2)=2.d0*x(2)
      gradgj(3)=2.d0*x(3)
      gradgj(4)=2.d0*x(4)
      return
      end
```

After running the algorithm on a SUN 4/SPARC station 1, the following output is obtained

```
            FSQP Version 3.0c (Released September 1992)
                    Copyright (c) 1989 --- 1992
                      J.L. Zhou and A.L. Tits
                        All Rights Reserved


        The given initial point is feasible for inequality
               constraints and linear equality constraints:
                               0.10000000000000E+01
                               0.50000000000000E+01
                               0.50000000000000E+01
                               0.10000000000000E+01

        iteration                                     8
        inform                                        0
        x                      0.10000000000000E+01
                               0.47429996518112E+01
                               0.38211499651796E+01
                               0.13794082958030E+01
        objectives             0.17014017289158E+02
        constraints           -0.35171865420125E-11
                              -0.35100811146549E-11
        SCV                    0.35100811146549E-11
        d0norm                 0.23956399867788E-07
        ktnorm                 0.34009891628142E-07
        ncallf                                        9
        ncallg                                       24


        Normal termination: You have obtained a solution !!
```

9 Results for Test Problems

These results are provided to allow the user to compare FSQP with his/her favorite code (see also [2–4]). Table 1 contains results obtained for some (non-minimax) test problems from [9] (the same initial points as in [9] were selected). `prob` indicates the problem number as in [9], `nineqn` the number of nonlinear constraints, `ncallf` the total number of evaluations of the objective function, `ncallg` the total number of evaluations of the (scalar) nonlinear constraint functions, `iter` the total number of iterations, `objective` the final value of the objective, `ktnorm` the norm of Kuhn-Tucker vector at the final iterate. `eps` the norm requirement of the Kuhn-Tucker vector, `SCV` the sum of feasibility violation of linear constraints (see §4). On each test problem, `eps` was selected so as to achieve the same field precision as in [9]. Whether FSQP-AL (0) or FSQP-NL (1) is used is indicated in column "B".

Results obtained on selected minimax problems are summarized in Table 2. Problems `bard`, `davd2`, `f&r`, `hettich`, and `wats` are from [11]; `cb2`, `cb3`, `r-s`, `wong` and `colv` are from [12; Examples 5.1-5] (the latest test results on problems `bard` down to `wong` can be found in [13]); `kiw1` and `kiw4` are from [14] (results for `kiw2` and `kiw3` are not reported due to data disparity); `mad1` to `mad8` are from [10, Examples 1-8]; `polk1` to `polk4` are from [15]. Some of these test problems allow one to freely select the number of variables; problems `wats-6` and `wats-20` correspond to 6 and 20 variables respectively, and `mad8-10`, `mad8-30` and `mad8-50` to 10, 30 and 50 variables respectively. All of the above are either unconstrained or linearly constrained minimax problems. Unable to find nonlinearly constrained minimax test problems in the literature, we constructed problems `p43m` through `p117m` from problems 43, 84, 113 and 117 in [9] by removing certain constraints and including instead additional objectives of the form

$$f_i(x) = f(x) + \alpha_i g_i(x)$$

where the α_i's are positive scalars and $g_i(x) \leq 0$. Specifically, `p43m` is constructed from problem 43 by taking out the first two constraints and including two corresponding objectives with $\alpha_i = 15$ for both; `p84m` similarly corresponds to problem 84 without constraints 5 and 6 but with two corresponding additional objectives, with $\alpha_i = 20$ for both; for `p113m`, the first three linear constraints from problem 113 were turned into objectives, with $\alpha_i = 10$ for all; for `p117m`, the first two nonlinear constraints were turned into objectives, again with $\alpha_i = 10$ for both. The gradients of all the functions were computed by finite difference approximation except for `polk1` through `polk4` for which gradients were computed analytically.

In Table 2, the meaning of columns `B`, `nineqn`, `ncallf`, `ncallg`, `iter`, `ktnorm` and `SCV` is as in Table 1 (but `ncallf` is the total number of evaluations of *scalar* objective function). `nf` is the number of objective functions in the max, `objmax` is the final value of the max of the objective functions. Finally, as in Table 1, `eps` is the stopping rule parameter. Here however its specific meaning varies from problem to problem as we attempted to best approximate the stopping rule used in the reference. Specifically, for problems `bard` through `kiw4`, execution was terminated when $\|d_k^0\|$ becomes smaller than the corresponding value of ϵ in the column of `eps` (this was also done for problems `p43m` through `p117m`); for problems `mad1` down to `mad8`, execution was terminated when $\|d_k^0\|$ is smaller than $\|x_k\|$ times the corresponding value of ϵ in the column `eps` (except `mad2` for which FSQPD was terminated when the 14 digits of the maximum objective value carried out by our code did not change); for problems `polk1` through `polk4`, execution was terminated when $\log_\epsilon \|x_k - x^*\|$ becomes smaller than the corresponding value of ϵ in the column of `eps`. FSQPD with monotone line search failed to reach a solution for `mad8-30` when QLD was used, but it succeeded when QPSOL [16] was used.[5]

Table 3 contains results of problems with nonlinear equality constraints from [9]. All symbols are the same as described before. `eps` is the norm requirement on d_k^0 and `epseqn` is chosen close to the corresponding values in [9], with 10^{-8} replacing 0. An asterisk (*) indicates that FSQP failed to meet the stopping criterion before certain execution error is encountered. It can be checked that the second order sufficient conditions of optimality are not satisfied at the known optimal solution for problems 26, 27, 46 and 47.

10 Limitations

It is important to keep in mind some limitations of FSQP. First, similar to most codes targeted at smooth problems, it is likely to encounter difficulties when confronted to non-smooth functions such as, for example, functions involving matrix eigenvalues. Second, because FSQP generates feasible iterates, it may be slow if the feasible set is very "thin" or oddly shaped. Third, if $h_j(x) \geq 0$ for all $x \in R^n$ and if $h_j(x_0) = 0$ for some j at the initial point x_0, the interior of the feasible set defined by $h_j(x) \leq 0$ for such j is empty. This may cause difficulties for FSQPD because, in FSQPD, $h_j(x) \geq 0$ is directly turned into $h_j(x) \leq 0$ for such j. The user is advised to either give an initial point that is infeasible for all nonlinear equality constraints or change the sign of h_j so that $h_j(x) < 0$ can be achieved at some point for all such nonlinear equality constraint.

Acknowledgment

The authors are indebted to Dr. E.R. Panier for many invaluable comments and suggestions.

References

[1] D.Q. Mayne & E. Polak, "Feasible Directions Algorithms for Optimization Problems with Equality and Inequality Constraints," *Math. Programming* 11 (1976) , 67–80.

[2] E.R. Panier & A.L. Tits, "On Combining Feasibility, Descent and Superlinear Convergence in Inequality Constrained Optimization," *Math. Programming* (1993, to appear) .

[5] But on most problems, according to our experience, QLD is significantly faster than QPSOL. A subroutine to interface FSQP with QPSOL can be obtained from the authors.

[3] J.F. Bonnans, E.R. Panier, A.L. Tits & J.L. Zhou, "Avoiding the Maratos Effect by Means of a Nonmonotone Line Search. II. Inequality Constrained Problems -- Feasible Iterates," *SIAM J. Numer. Anal.* 29 (1992) , 1187–1202.

[4] J.L. Zhou & A.L. Tits, "Nonmonotone Line Search for Minimax Problems," *J. Optim. Theory Appl.* 76 (March 1993, to appear) .

[5] L. Grippo, F. Lampariello & S. Lucidi, "A Nonmonotone Line Search Technique for Newton's Method," *SIAM J. Numer. Anal.* 23 (1986) , 707–716.

[6] D. Q. Mayne & E. Polak, "A Superlinearly Convergent Algorithm for Constrained Optimization Problems," *Math. Programming Stud.* 16 (1982) , 45–61.

[7] K. Schittkowski, *QLD : A FORTRAN Code for Quadratic Programming, User's Guide*, Mathematisches Institute, Universität Bayreuth, Germany, 1986.

[8] M.J.D. Powell, "A Fast Algorithm for Nonlinearly Constrained Optimization Calculations," in *Numerical Analysis, Dundee, 1977, Lecture Notes in Mathematics 630*, G.A. Watson, ed., Springer-Verlag, 1978, 144–157.

[9] W. Hock & K. Schittkowski, *Test Examples for Nonlinear Programming Codes*, Lecture Notes in Economics and Mathematical Systems (187), Springer Verlag, 1981.

[10] K. Madsen & H. Schjær-Jacobsen, "Linearly Constrained Minimax Optimization," *Math. Programming* 14 (1978) , 208–223.

[11] G.A. Watson, "The Minimax Solution of an Overdetermined System of Non-linear Equations," *J. Inst. Math. Appl.* 23 (1979) , 167–180.

[12] C. Charalambous & A.R. Conn, "An Efficient Method to Solve the Minimax Problem Directly," *SIAM J. Numer. Anal.* 15 (1978) , 162–187.

[13] A.R. Conn & Y. Li, "An Efficient Algorithm for Nonlinear Minimax Problems," University of Waterloo, Research Report CS-88-41, Waterloo, Ontario, N2L 3G1 Canada, November, 1989 .

[14] K.C. Kiwiel, *Methods of Descent in Nondifferentiable Optimization*, Lecture Notes in Mathematics #1133, Springer-Verlag, Berlin, Heidelberg, New-York, Tokyo, 1985.

[15] E. Polak, D.Q. Mayne & J.E. Higgins, "A Superlinearly Convergent Algorithm for Minmax Problems," *Proceedings of the 28th IEEE Conference on Decision and Control* (December 1989) .

[16] P.E. Gill, W. Murray, M.A. Saunders & M.H. Wright. "User's Guide for SOL/QPSOL: A FORTRAN Package for Quadratic Programming," Stanford Univ., Technical Report SOL 83-7, 1983.

prob	B	nineqn	ncallf	ncallg	iter	objective	ktnorm	eps	SCV
p12	0	1	7	14	7	−.300000000E+02	.72E-06	.10E-05	.0
	1		7	12	7	−.300000000E+02	.79E-06	.10E-05	.0
p29	0	1	11	20	10	−.226274170E+02	.41E-05	.10E-04	.0
	1		12	16	12	−.226274170E+02	.63E-05	.10E-04	.0
p30	0	1	13	25	13	.100000000E+01	.26E-07	.10E-06	.0
	1		14	14	14	.100000000E+01	.43E-07	.10E-06	.0
p31	0	1	10	21	8	.600000000E+01	.34E-06	.10E-04	.0
	1		10	18	10	.600000000E+01	.50E-06	.10E-04	.0
p32	0	1	3	5	3	.100000000E+01	.15E-14	.10E-07	.0
	1		3	4	3	.100000000E+01	.64E-16	.10E-07	.0
p33	0	2	4	11	4	−.400000000E+01	.13E-11	.10E-07	.0
	1		5	10	5	−.400000000E+01	.47E-11	.10E-07	.0
p34	0	2	7	28	7	−.834032443E+00	.19E-08	.10E-07	.0
	1		9	24	9	−.834032445E+00	.38E-11	.10E-07	.0
p43	0	3	11	51	9	−.440000000E+02	.12E-05	.10E-04	.0
	1		12	49	12	−.440000000E+02	.16E-06	.10E-04	.0
p44	0	0	6	0	6	−.150000000E+02	.0	.10E-07	.0
	1		6		6	−.150000000E+02	.0	.10E-07	.0
p51	0	0	8	0	6	.505655658E−15	.46E-06	.10E-05	.22E-15
	1		9		8	.505655658E−15	.34E-08	.10E-05	.22E-15
p57	0	1	5	7	3	.306463061E−01	.29E-05	.10E-04	.0
	1		5	7	3	.306463061E−01	.28E-05	.10E-04	.0
p66	0	2	8	30	8	.518163274E+00	.50E-09	.10E-07	.0
	1		9	24	9	.518163274E+00	.14E-08	.10E-07	.0
p67	0	14	21	305	21	−.116211927E+02	.88E-06	.10E-04	.0
	1		61	854	61	−.116211927E+02	.58E-05	.10E-04	.0
p70	0	1	32	39	30	.940197325E−02	.58E-08	.10E-06	.0
	1		31	31	31	.940197325E−02	.19E-07	.10E-06	.0
p76	0	0	6	0	6	−.468181818E+01	.34E-04	.10E-03	.0
	1		6		6	−.468181818E+01	.34E-04	.10E-03	.0
p84	0	6	4	30	4	−.528033513E+07	.0	.10E-07	.0
	1		4	29	4	−.528033513E+07	.38E-09	.10E-07	.0
p85	0	38	34	1347	34	−.240000854E+01	.35E-03	.10E-02	.0
	1		80	3040	80	−.240000854E+01	.81E-03	.10E-02	.0
p86	0	0	8	0	6	−.323486790E+02	.22E-08	.10E-05	.0
	1		7		6	−.323486790E+02	.53E-06	.10E-05	.0
p93	0	2	15	58	12	.135075968E+03	.37E-03	.10E-02	.0
	1		15	36	15	.135075964E+03	.24E-04	.10E-02	.0
p100	0	4	23	114	16	.680630057E+03	.62E-06	.10E-03	.0
	1		20	102	17	.680630057E+03	.49E-04	.10E-03	.0
p110	0	0	9	0	8	−.457784697E+02	.50E-06	.10E-05	.0
	1		9		8	−.457784697E+02	.50E-06	.10E-05	.0
p113	0	5	12	108	12	.243063768E+02	.81E-03	.10E-02	.0
	1		12	99	12	.243064357E+02	.83E-03	.10E-02	.35E-14
p117	0	5	20	219	19	.323486790E+02	.58E-04	.10E-03	.0
	1		18	93	17	.323486790E+02	.34E-04	.10E-03	.0
p118	0	0	19	0	19	.664820450E+03	.13E-14	.10E-07	.0
	1		19		19	.664820450E+03	.17E-14	.10E-07	.0

Table 1: Results for Inequality Constrained Problems with FSQP Version 3.0c

prob	B	nineqn	nf	ncallf	ncallg	iter	objmax	ktnorm	eps	SCV
bard	0	0	15	168	0	8	.508163265E−01	.61E-09	.50E-05	.0
	1			105		7	.508168686E−01	.22E-06	.50E-05	.0
cb2	0	0	3	30	0	6	.195222449E+01	.37E-06	.50E-05	.0
	1			18		6	.195222449E+01	.29E-05	.50E-05	.0
cb3	0	0	3	15	0	3	.200000157E+01	.40E-05	.50E-05	.0
	1			15		5	.200000000E+01	.47E-08	.50E-05	.0
colv	0	0	6	240	0	21	.323486790E+02	.46E-05	.50E-05	.0
	1			102		17	.323486790E+02	.12E-04	.50E-05	.0
davd2	0	0	20	342	0	12	.115706440E+03	.62E-06	.50E-05	.0
	1			220		11	.115706440E+03	.11E-05	.50E-05	.0
f&r	0	0	2	32	0	9	.494895210E+01	.90E-09	.50E-05	.0
	1			20		10	.494895210E+01	.70E-07	.50E-05	.0
hettich	0	0	5	125	0	13	.245935695E−02	.10E-07	.50E-05	.0
	1			75		11	.245936698E−02	.18E-07	.50E-05	.0
r-s	0	0	4	71	0	9	−.440000000E+02	.98E-06	.50E-05	.0
	1			68		12	−.440000000E+02	.28E-06	.50E-05	.0
wats-6	0	0	31	623	0	12	.127172748E−01	.42E-07	.50E-05	.0
	1			433		13	.127170913E−01	.84E-10	.50E-05	.0
wats-20	0	0	31	1953	0	32	.895554035E−07	.13E-05	.50E-05	.0
	1			1023		32	.898278737E−07	.13E-05	.50E-05	.0
wong	0	0	5	182	0	19	.680630057E+03	.40E-04	.50E-05	.0
	1			171		26	.680630057E+03	.13E-03	.50E-05	.0
kiw1	0	0	10	159	0	11	.226001621E+02	.32E-05	.11E-05	.0
	1			130		13	.226001621E+02	.54E-05	.60E-06	.0
kiw4	0	0	2	42	0	9	.222044605E−15	.18E-07	.42E-07	.0
	1			23		9	.0	.47E-07	.15E-07	.0
mad1	0	0	3	24	0	5	−.389659516E+00	.22E-10	.10E-09	.0
	1			18		6	−.389659516E+00	.48E-10	.10E-09	.0
mad2	0	0	3	25	0	5	−.330357143E+00	.22E-10	.10E-09	.0
	1			21		6	−.330357143E+00	.86E-09	.10E-09	.0
mad4	0	0	3	29	0	6	−.448910786E+00	.31E-17	.10E-09	.0
	1			24		8	−.448910786E+00	.38E-16	.10E-09	.0
mad5	0	0	3	31	0	7	−.100000000E+01	.21E-11	.10E-09	.0
	1			24		8	−.100000000E+01	.78E-14	.10E-09	.0
mad6	0	0	163	1084	0	6	.113104727E+00	.81E-11	.10E-09	.0
	1			1141		7	.113104727E+00	.21E-10	.10E-09	.0
mad8-10	0	0	18	291	0	10	.381173963E+00	.89E-11	.10E-09	.0
	1			252		14	.381173963E+00	.16E-14	.10E-09	.0
mad8-30	0	0				*			.10E-09	
	1			1102		18	.547620496E+00	.12E-14	.10E-09	.0
mad8-50	0	0	98	3056	0	21	.579276202E+00	.86E-15	.10E-09	.0
	1			2084		21	.579276202E+00	.91E-16	.10E-09	.0
polk1	0	0	2	42	0	10	.271828183E+01	.50E-04	−10.00	.0
	1			22		10	.271828183E+01	.68E-04	−10.00	.0
polk2	0	0	2	203	0	42	.545981839E+02	.28E-03	− 9.00	.0
	1			116		38	.545981500E+02	.14E-02	− 9.00	.0
polk3	0	0	10	188	0	12	.370348302E+01	.23E-02	− 5.50	.0
	1			141		12	.370348272E+01	.26E-02	− 5.50	.0
polk4	0	0	3	45	0	7	.0	.39E-04	−10.00	.0
	1			24		7	.364604254E+00	.37E-06	−10.00	.0
p43m	0	1	3	80	43	15	−.440000000E+02	.14E-05	.50E-05	.0
	1			63	25	16	−.440000000E+02	.46E-05	.50E-05	.0
p84m	0	4	3	17	20	4	−.528033513E+07	.28E-09	.50E-05	.0
	1			9	12	3	−.528033511E+07	.76E-05	.50E-05	.0
p113m	0	5	4	108	127	14	.243062091E+02	.14E-04	.50E-05	.0
	1			84	105	14	.243062091E+02	.29E-04	.50E-05	.0
p117m	0	3	3	124	144	21	.323486790E+02	.43E-05	.50E-05	.0
	1			57	54	17	.323486790E+02	.26E-04	.50E-05	.0

Table 2: Results for Minimax Problems with FSQP Version 3.0c

prob	B	ncallf	ncallg	iter	objective	ktnorm	eps	epseqn	SCV
p6	0	17	22	10	.274055126E−11	.42E-05	.10E-03	.40E-06	.20E-09
	1	21	23	10	.116074629E−12	.35E-05	.10E-03	.40E-06	.28E-06
p7	0	57	57	13	−.173205081E+01	.12E-06	.10E-03	.35E-08	.70E-09
	1	27	25	15	−.173205081E+01	.68E-08	.10E-03	.35E-08	.15E-09
p26	0	127	138	51	.270576724E−13	.15E-08	.10E-03	.16E-04	.12E-09
	1	38	38	31	.322181110E−13	.49E-08	.10E-03	.16E-04	.43E-08
p27	0	153	147	44	.399986835E−01	.24E-02	.10E-02	.10E-02	.38E-04
	1	999	996	130	.399916645E−01	.39E-03	.10E-02	.10E-02	.21E-03
p39	0	23	49	17	−.100000000E+01	.39E-04	.10E-03	.75E-04	.90E-08
	1	12	25	12	−.100000000E+01	.50E-04	.10E-03	.75E-04	.64E-06
p40	0	5	15	5	−.250000002E+01	.26E-05	.10E-03	.85E-04	.96E-08
	1	5	17	5	−.250000000E+01	.41E-04	.10E-03	.85E-04	.43E-05
p42	0	9	10	6	.138578644E+02	.27E-05	.10E-03	.45E-05	.51E-09
	1	7	12	7	.138578652E+02	.26E-03	.10E-03	.45E-05	.33E-06
p46	0	62	135	26	.224262538E−10	.11E-04	.10E-03	.50E-04	.57E-10
	1	56	25	14	.461984187E−04	.19E-02	.10E-03	.50E-04	.95E-06
p47	0	74	241	38	.162241544E−11	.56E-06	.10E-03	.60E-04	.41E-09
	1	50	282	36	.308185534E−01	.11E-04	.10E-03	.60E-04	.26E-08
p56	0	31	147	15	−.345600000E+01	.46E-08	.10E-03	.25E-06	.34E-10
	1	14	60	14	−.345600000E+01	.88E-05	.10E-03	.25E-06	.11E-08
p60	0	10	13	10	.325682003E−01	.29E-05	.10E-03	.55E-04	.27E-09
	1	9	14	9	.325687946E−01	.21E-03	.10E-03	.55E-04	.55E-04
p61	0	18	38	8	−.143646142E+03	.35E-04	.10E-03	.25E-06	.13E-07
	1	38	17	9	−.143646142E+03	.67E-07	.10E-03	.25E-06	.27E-12
p63	0	8	10	8	.961715172E+03	.12E-06	.10E-03	.60E-05	.15E-10
	1	6	10	6	.961715172E+03	.25E-04	.10E-03	.60E-05	.65E-07
p71	0	9	24	8	.170140173E+02	.34E-07	.10E-03	.70E-05	.35E-11
	1	6	19	6	.170140173E+02	.79E-09	.10E-03	.70E-05	.28E-08
p74	0	14	43	14	.512649811E+04	.65E-06	.10E-03	.65E-05	.21E-10
	1	41	123	41	.512649811E+04	.31E-04	.10E-03	.65E-05	.16E-08
p75	0	13	39	13	.517441270E+04	.84E-08	.10E-03	.10E-07	.25E-11
	1	28	84	28	.517441270E+04	.35E-08	.10E-03	.10E-07	.19E-08
p77	0	15	37	15	.241505129E+00	.30E-05	.10E-03	.35E-04	.68E-07
	1	18	48	19	.241505211E+00	.61E-04	.10E-03	.35E-04	.14E-05
p78	0	9	41	9	−.291970041E+01	.83E-07	.10E-03	.15E-05	.45E-10
	1	8	26	8	−.291970041E+01	.11E-03	.10E-03	.15E-05	.11E-08
p79	0	7	24	7	.974340336E−01	.12E-04	.10E-03	.15E-03	.41E-07
	1	10	34	10	.974340336E−01	.66E-05	.10E-03	.15E-03	.40E-07
p80	0	66	198	20	.539498478E−01	.25E-08	.10E-03	.15E-07	.25E-12
	1	7	21	7	.539498478E−01	.91E-08	.10E-03	.15E-07	.11E-07
p81	0	59	177	20	.539498478E−01	.55E-05	.10E-03	.80E-06	.36E-09
	1	8	24	8	.539498419E−01	.63E-05	.10E-03	.80E-06	.17E-06
p99	0	111	269	38	−.831079886E+09	.17E+03	.10E-03	.10E-07	.92E-09
	1	130	1229	130	−.831079886E+09	.33E+01	.10E-03	.10E-07	.50E-01
p107	0	16	116	14	.505501180E+04	.56E-02	.10E-03	.10E-07	.48E-09
	1	16	109	16	.505501180E+04	.69E-03	.10E-03	.10E-07	.39E-09
p109	0			*			.10E-03	.10E-07	
	1			*			.10E-03	.10E-07	
p114	0			*			.10E-02	.10E-03	
	1	18241	941	924	−.176880696E+04	.51E-05	.10E-02	.10E-03	.22E-12

Table 3: Results for General Problems with FSQP Version 3.0c