TECHNICAL RESEARCH REPORT

SYSTEMS RESEARCH CENTER

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

User's Guide for FSQP Version 2.0 A Fortran Code for Solving Optimization Problems, Possibly Minimax, with General Inequality Constraints and Linear Equality Constraints, Generating Feasible Iterates

by J. Zhou and A.L. Tits

TR 90-60r1

User's Guide for FSQP Version 2.0 A Fortran Code for Solving Optimization Problems, Possibly Minimax, with General Inequality Constraints and Linear Equality Constraints, Generating Feasible Iterates[1]

Jian Zhou and André L. Tits

Electrical Engineering Department

and

Systems Research Center

University of Maryland, College Park, MD 20742

Abstract

FSQP 2.0 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 smooth inequality constraints, linear inequality and linear equality constraints, and simple bounds on the variables. If the initial guess provided by the user is infeasible, FSQP first generates a feasible point; subsequently the successive iterates generated by FSQP all satisfy the constraints. The user has the option of requiring that the maximum value among the objective functions decrease at each iteration after feasibility has been reached (monotone 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 2.0 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.

[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 for commercial applications, unless this has been agreed upon with the authors in writing.

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@cacse.src.umd.edu

Contents

1 Introduction

FSQP (Feasible Sequential Quadratic Programming) 2.0 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 inequality constraints, linear inequality and equality constraints, and simple bounds on the variables. Specifically, FSQP tackles optimization problems of the form

$$(P) \qquad \min \max_{i=1,\ldots,n_f} \{f_i(x)\} \quad \text{s.t.} \quad x \in X$$

where X is the set of point $x \in R^n$ satisfying

$$\begin{array}{l} bl \leq x \leq bu \\ g_j(x) \leq 0, \quad j = 1,\ldots,n_i \\ g_j(x) \equiv \langle c_{j-n_i}, x\rangle - d_{j-n_i} \leq 0, \quad j = n_i+1,\ldots,t_i \\ \langle a_j, x\rangle = b_j \quad j = 1,\ldots,l_e \end{array}$$

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

If the initial guess provided by the user is infeasible, FSQP first generates a feasible point. Subsequently the successive iterates generated by FSQP all satisfy the constraints. The user has the option of requiring that the maximum value of the objective functions decrease at each iteration after feasibility has been reached. 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 2.0 implements two algorithms that are described and analyzed in [1] and [2], with some additional refinements. These algorithms are based on a Sequential Quadratic Programming (SQP) iteration, modified so as to generate feasible iterates. The SQP direction is first "tilted" 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 merit function used in this arc search is the maximum of the objective functions. An Armijo-type line search is always selected 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 maximum of the objective functions at each iteration[1]; or (ii) a nonmonotone line search (inspired from [3] and analyzed in [2] in this context) may be selected, forcing a decrease of the maximum value of the objective functions within at most four iterations. The latter 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.

FSQP 2.0 invokes the quadratic programming routine QPSOL [4] which the user must provide.

2 Description of the Algorithms

The algorithms described and analyzed in [1] and [2] are as follows. In both algorithms, 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 and is almost feasible in the sense that it is at worst tangent to the feasible set.

In [1], 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 $\tilde{d} = \tilde{d}(x,H)$ 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 [1] on $d^1(\cdot)$, $\rho(\cdot)$ and $\tilde{d}(\cdot,\cdot)$ that result in a globally convergent, locally superlinear convergent algorithm.

The algorithm in [2] 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 $\tilde{d} = \tilde{d}(x,H)$ is computed the same way as in [1], 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 [2] 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.

The FSQP implementation corresponds to a specific choice of these functions, 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 details are given below. The analysis in [1] and [2] can be easily extended to these modified algorithms. Also obvious simplifications are introduced concerning the linear inequality constraints, and linear equality constraints are allowed as well: the iterates are allowed (resp. forced) 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

$$f(x) = \max_{i=1,\ldots,n_f} \{f_i(x)\}$$

$$f'(x,d) = \max_{i=1,\ldots,n_f} \{f_i(x) + \langle \nabla f_i(x), d\rangle\} - f(x)$$

and

$$\tilde{f}'(x+d, x, \tilde{d}) = \max_{i=1,\ldots,n_f} \{f_i(x+d) + \langle \nabla f_i(x), \tilde{d}\rangle\} - f(x+d).$$

Algorithm FSQP-AL.

Parameters. $\eta = 0.1$, $\nu = 0.01$, $\alpha = 1.0 \times 10^{-7}$, $\beta = 0.5$, $\kappa = 2.1$, $\tau_1 = \tau_2 = 2.5$.

Data. $x_0 \in R^n$, $\varepsilon > 0$.

Step 0: Initialization. Set $k = 0$ and $H_0 = I$, the identity matrix. If x_0 is infeasible, substitute a feasible point, obtained as discussed below.

Step 1: Computation of a search arc.

i. Compute d_k^0 by solving the strictly convex quadratic program

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

Compute the Kuhn-Tucker vector

$$\nabla L(x_k, \zeta_k, \xi_k, \lambda_k, \mu_k) = \sum_{j=1}^{n_f} \zeta_{k,j} \nabla f_j(x_k) + \sum_{j=1}^{n} \xi_{k,j} + \sum_{j=1}^{t_i} \lambda_{k,j} \nabla g_j(x_k) + \sum_{j=1}^{l_e} \mu_{k,j} a_j$$

where the $\zeta_{k,j}$'s with $\sum_{j=1}^{n_f} \zeta_{k,j} = 1$, $\xi_{k,j}$'s, $\lambda_{k,j}$'s, and $\mu_{k,j}$'s are 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.

If $\|\nabla L(x_k, \zeta_k, \xi_k, \lambda_k, \mu_k)\| \le \varepsilon$, stop. If $n_i = 0$ and $n_f = 1$, set $d_k = d_k^0$ and $\tilde{d}_k = 0$ and go to *Step 2*. If $n_i = 0$ 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

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

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

$$\begin{array}{ll} \min\limits_{\tilde{d} \in R^n} & \frac{1}{2}\langle (d_k + \tilde{d}), H_k(d_k + \tilde{d})\rangle + f'(x_k, d_k, \tilde{d}) \\ \text{s.t.} & bl \le x_k + d_k + \tilde{d} \le bu \\ & g_j(x_k + d_k) + \langle \nabla g_j(x_k), \tilde{d}\rangle \le -\min(\nu\|d_k\|, \|d_k\|^{\tau_2}), \quad j = 1, \ldots, n_i \\ & \langle c_j, x_k + d_k + \tilde{d}\rangle \le d_j, \quad j = 1, \ldots, t_i - n_i \\ & \langle a_j, x_k + d_k + \tilde{d}\rangle = b_j, \quad j = 1, \ldots, l_e \end{array}$$

where $f'(x_k, d_k, \tilde{d}) = f'(x_k, d_k + \tilde{d})$ if $n_f = 1$, and $f'(x_k, d_k, \tilde{d}) = \tilde{f}'(x_k + d_k, x_k, \tilde{d})$ 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. Compute t_k, the first number t in the sequence $\{1, \beta, \beta^2, \ldots\}$ satisfying

$$f(x_k + td_k + t^2\tilde{d}_k) \le f(x_k) + \alpha t f'(x_k, d_k)$$

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

The line search is actually performed in such a way that those functions with nonzero multipliers in the QP yielding d_k^0 will be evaluated first; only if all of them satisfy the line search rule will the rest of the functions be tested.

Step 3. Updates.

- Compute a new approximation H_{k+1} to the Hessian of the Lagrangian using the BFGS formula with Powell's modification[5].

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

- Increase k by 1.
- Go back to *Step 1.*

$\square$

Algorithm FSQP-NL.

Parameters. $\eta = 3.0$, $\nu = 0.01$, $\alpha = 1.0 \times 10^{-7}$, $\beta = 0.5$, $\theta = 0.2$, $\bar{\rho} = 0.5$, $\gamma = 2.5$, $\underline{C} = 0.01$, $\underline{d} = 5.0$.

Data. $x_0 \in R^n$, $\varepsilon > 0$.

Step 0: Initialization. Set $k = 0$, $H_0 = I$, the identity matrix, and $C_0 = \underline{C}$. If x_0 is infeasible, substitute a feasible point, obtained as discussed below. Set $x_{-3} = x_{-2} = x_{-1} = x_0$.

Step 1: Computation of a new iterate.

i. Compute d_k^0 by solving the strictly convex quadratic program

$$\begin{array}{ll} \min\limits_{d^0 \in R^n} & \frac{1}{2}\langle d^0, H_k d^0\rangle + f'(x_k, d^0) \\ \text{s.t.} & bl \leq x_k + d^0 \leq bu \\ & g_j(x_k) + \langle \nabla g_j(x_k), d^0\rangle \leq 0, \quad j = 1, \ldots, t_i \\ & \langle a_j, x_k + d^0\rangle = b_j, \quad j = 1, \ldots, l_e \end{array}$$

Compute the Kuhn-Tucker vector

$$\nabla L(x_k, \zeta_k, \xi_k, \lambda_k, \mu_k) = \sum_{j=1}^{n_f} \zeta_{k,j} \nabla f_j(x_k) + \sum_{j=1}^{n} \xi_{k,j} + \sum_{j=1}^{t_i} \lambda_{k,j} \nabla g_j(x_k) + \sum_{j=1}^{l_e} \mu_{k,j} a_j$$

where the $\zeta_{k,j}$'s with $\sum_{j=1}^{n_f} \zeta_{k,j} = 1$, $\xi_{k,j}$'s, $\lambda_{k,j}$'s, and $\mu_{k,j}$'s are the multipliers, for the various functions, simple bounds (only n possible active bounds at each iteration), inequality, and equality constraints respectively, associated with this quadratic program.

If $\|\nabla L(x_k, \zeta_k, \xi_k, \lambda_k, \mu_k)\| \leq \varepsilon$, stop. If $n_i = 0$ and $n_f = 1$, set $d_k = d_k^0$ and $\tilde{d}_k = 0$ and go to *Step 1 viii.* If $n_i = 0$ 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

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

iii. Set $v_k = \min\{C_k\|d_k^0\|^2, \|d_k^0\|\}$. Define values $\rho_{k,j}$ for $j = 1, \ldots, n_i$ by $\rho_{k,j}$ 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. Let $\rho_k^\ell = \max_{j=1,\ldots,n_i}\{\rho_{k,j}\}$.

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

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

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

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

If

$$f(x_k + d_k^\ell) \leq \max_{l=0,\ldots,3}\{f(x_{k-l})\} + \alpha f'(x_k, d_k^0)$$

and

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

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

$$\begin{array}{ll}
\min\limits_{\tilde{d}\in R^n} & \frac{1}{2}\langle(d_k^g + \tilde{d}), H_k(d_k^g + \tilde{d})\rangle + f'(x_k, d_k^g, \tilde{d}) \\
\text{s.t.} & bl \leq x_k + d_k^g + \tilde{d} \leq bu \\
& g_j(x_k + d_k^g) + \langle \nabla g_j(x_k), \tilde{d} \rangle \leq -\min(\nu\|d_k^g\|,\ \|d_k^g\|^\tau), \quad j = 1, \ldots, n_i \\
& \langle c_j, x_k + d_k^g + \tilde{d} \rangle \leq d_j, \quad j = 1, \ldots, t_i - n_i \\
& \langle a_j, x_k + d_k^g + \tilde{d} \rangle = b_j, \quad j = 1, \ldots, l_e
\end{array}$$

where $f'(x_k, d_k^g, \tilde{d}) = f'(x_k, d_k^g + \tilde{d})$ if $n_f = 1$, and $f'(x_k, d_k^g, \tilde{d}) = \tilde{f}'(x_k + d_k^g, x_k, \tilde{d})$ 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. Compute t_k, the first number t in the sequence $\{1, \beta, \beta^2, \ldots\}$ satisfying

$$f(x_k + td_k^g + t^2\tilde{d}_k) \leq \max_{l=0,\ldots,3}\{f(x_{k-l})\} + \alpha t f'(x_k, d_k^g)$$

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

and set $x_{k+1} = x_k + t_k d_k^g + t_k^2 \tilde{d}_k$.
The line search is actually performed in such a way that those functions with nonzero multipliers in the QP yielding d_k^0 will be evaluated first; only if all of them satisfy the line search rule will the rest of the functions be tested.

Step 2. Updates.

- Compute a new approximation H_{k+1} to the Hessian of the Lagrangian using the BFGS formula with Powell's modification[5].
- 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 $C_{k+1} = 10C_k$.
- Increase k by 1.
- Go back to *Step 1.*

$\square$

If the initial guess x_0 provided by the user is not feasible, FSQP first solves a strictly convex quadratic program

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

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

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

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

3 Specification of Subroutine FSQPD 2.0

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

```
      subroutine FSQPD(nparam,nf,Linfty,nnl,nineq,neq,mode,iprint,
     *                 miter,inform,bigbnd,eps,udelta,bl,bu,x,f,g,
     *                 iw,iwsize,w,nwsize,obj,constr,gradob,gradcn)
      implicit double precision (a-h,o-z)
      dimension bl(nparam),bu(nparam),x(nparam),f(nf),
     *          g(nineq+neq),iw(iwsize),w(nwsize)
      external obj,constr,gradob,gradcn
      logical Linfty
```

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

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

`Linfty` **(Input)** Logical variable with `Linfty=.false.` if (P) is to be solved, and `Linfty=.true.` if (PL_∞) is to be solved. (PL_∞) is defined as follows

$$(PL_\infty) \qquad \min \max_{i=1,\ldots,n_f} |f_i(x)| \quad \text{s.t.} \quad x \in X$$

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 $i = 1, \ldots, n_f$.

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

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

`neq` **(Input)** (integer) Number (possibly zero) of linear equality constraints (l_e in the algorithm description).

`mode` **(Input)** Integer variable:

- `mode` $= 0$: Algorithm FSQP-AL is selected, resulting in a decrease of the maximum value of the objective functions at each iteration.
- `mode` $= 1$: Algorithm FSQP-NL is selected, resulting in a decrease of the maximum value of the objective functions within at most four iterations.

`iprint` **(Input)** Integer variable indicating the desired output (see Section 4 for details):

- `iprint` $= 0$: No information except for user-input errors is displayed.
- `iprint` $= 1$: At the end of execution, status (`inform`), iterate, objective value, constraint values, number of evaluations of objective 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)** (integer) Maximum number of iterations allowed by the user before termination of execution.

`inform` **(Output)** Integer indicating the results of FSQPD.

- `inform` $= 0$: Normal termination of execution in the sense that the norm of the final Kuhn-Tucker vector $\nabla L(x, \zeta, \xi, \lambda, \mu)$ is no greater than `eps`.
- `inform` $= 1$: The user-provided initial guess is infeasible and FSQPD is unable to generate a point satisfying all linear constraints.
- `inform` $= 2$: The user-provided initial guess is infeasible and FSQPD is unable to generate a point satisfying all constraints.

`inform` = 3 :	The maximum number `miter` of iterations has been reached before a solution is obtained.
`inform` = 4 :	The line search fails to find a new iterate (trial step size being smaller than the machine precision `epsmac` computed by FSQPD).
`inform` = 5 :	Failure in attempting to construct d^0.
`inform` = 6 :	Failure in attempting to construct d^1.
`inform` = 7 :	Input data are not consistent (with printout indicating the error).

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

`eps` **(Input)** (double precision) Final norm requirement for the Kuhn-Tucker vector (ε in the algorithm description). It must be bigger than the machine precision `epsmac` (computed by FSQPD).

`udelta` **(Input)** (double precision) The perturbation size the user suggests to use in approximating gradients by finite difference. The perturbation size actually used is defined by $\text{sign}(x^i) \times \max\{$`udelta`, `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)** (double precision) Array of dimension `nparam` containing lower bounds for the components of `x`. To specify a non-existent lower bound (i.e., `bl`$(j) = -\infty$ for some j), the value used must satisfy `bl`$(j) \leq -$`bigbnd`.

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

`x` **(Input)** (double precision) Initial guess.
(Output) Final solution if FSQP terminates with `inform=0`.

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

`g` (double precision) Array of dimension `nineq` + `neq`.
(Output) Values of constraints at `x` at the end of execution.

`iw` Integer workspace vector of dimension `iwsize`.

`iwsize` **(Input)** Integer workspace length for `iw`. It must be at least as big as $3 \times$ `nparam` $+ 3 \times \max(1,$ `nineq` $+$ `neq` $+$ `nfunc`$) +$ `nnl` $+$ `nf` $+ 6$. Here `nfunc` $=$ `nf` if `Linfty` $=$ `.false.` and `nfunc` $= 2\times$`nf` if `Linfty` $=$ `.true.`. The smallest suitable value will be displayed if the user-supplied value is too small.

`w` Double precision workspace of dimension `nwsize`.

`nwsize` **(Input)** Workspace length for `w`. It must be at least as big as $4 \times \mathtt{nparam}^2 + 2 \times \mathtt{nparam} \times \max(1, \mathtt{nineq} + \mathtt{neq}) + \mathtt{nparam} \times \mathtt{nf} + \mathtt{nparam} \times \mathtt{nfunc} + 20 \times \mathtt{nparam} + 10 \times \max(1, \mathtt{nineq} + \mathtt{neq} + \mathtt{nfunc}) + \mathtt{mm} + 22$. Here `nfunc` $=$ `nf` if `Linfty` $=$ `.false.` and `nfunc` $= 2\times$`nf` if `Linfty` $=$ `.true.`; and `mm` $=$ `M` if `mode` $= 1$ and `mm` $= 0$ otherwise. The smallest suitable value will be displayed if the user-supplied value is too small.

`obj` **(Input)** Name of the user-defined function 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 Section 5.1 below.

`constr` **(Input)** Name of the user-defined function 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 Section 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 Section 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 Section 5.4 below.

4 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 Section 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 `mode`$= 1$) Largest value of the maximum of the objective functions over the last 4 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 $1 \leq i \leq n_f$.

`ncallg` Number of evaluations (so far) of individual (scalar) nonlinear constraints.

`ktnorm` Norm of the Kuhn-Tucker vector at the current iteration. If execution terminates normally (`inform` $= 0$), then `ktnorm` $\leq$ `eps`.

`SCV` Sum of feasibility violation of linear constraints (since QPSOL does not accept zero tolerance of feasibility violation, there may be very small feasibility violation).

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:

`d0norm` Norm of the quasi-Newton direction d_k^0.

`d1norm` Norm of the first order direction d_k^1.

`d` (`mode=0`) Search direction $d_k = (1 - \rho_k)d_k^0 + \rho_k d_k^1$.

`dnorm` (`mode`=0) Norm of d_k.

`rho` (`mode`=0) Coefficient ρ_k in $d_k = (1-\rho_k)d_k^0 + \rho_k d_k^1$.

`dlnorm` (`mode`=1) Norm of d_k^ℓ.

`rhol` (`mode`=1) Coefficient ρ_k^ℓ in $d_k^\ell = (1-\rho_k^\ell)d_k^0 + \rho_k^\ell d_k^1$.

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

`rhog` (`mode`=1) Coefficient ρ_k^g in $d_k^g = (1-\rho_k^g)d_k^0 + \rho_k^g d_k^1$.

`dtilde` Correction $\tilde{d}_k$.

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

Details in the line search:

`trial step` The 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's and the corresponding values of the functions $f_i(x)$ for $1 \le i \le n_f$ up to the one which fails in line search at the `trial point`. The indices i's 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 i's and the corresponding values of nonlinear constraints for $1 \le i \le n_i$ up to the one which is not feasible at the `trial point`. The indices i's 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.

`multipliers` Multiplier estimates ordered as ξ's, λ's, μ's, and ζ's (from quadratic program computing d_k^0). $\lambda_i \ge 0 \quad \forall i = 1, \ldots, t_i$ and $\mu_i \ge 0 \quad \forall i = 1, \ldots, l_e$. $\xi_i > 0$ indicates that x_i reaches its upper bound and $\xi_i < 0$ indicates that x_i reaches its lower bound. When `Linfty` = `.false.`, $\zeta_i \ge 0$. When `Linfty` = `.true.`, $\zeta_i > 0$ refers to $+f_i(x)$ and $\zeta_i < 0$ to $-f_i(x)$.

`hess`	New estimate of the Hessian matrix of the Lagrangian.
`Ck`	The value C_k as defined in Algorithm FSQP-NL.

5 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.

5.1 Function obj

The function **obj**, to be provided by the user, computes the value of the objective functions. The specification of `obj` for FSQPD is

```
      function obj(nparam,j,x)
      implicit double precision (a-h,o-z)
      dimension x(nparam)
c
c     for given j, assign to obj 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.

5.2 Function constr

The function **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 f77 compiling requirement. The specification of `constr` for FSQPD is as follows

```
      function constr(nparam,j,x)
      implicit double precision (a-h,o-z)
      dimension x(nparam)
```

```
c
c     for given j, assign to constr 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.

The order of the constraints must be as follows. First the `nnl` (possibly zero) nonlinear inequality constraints. Then the `nineq-nnl` (possibly zero) linear inequality constraints. Finally, the `neq` (possibly zero) linear equality constraints.

5.3 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). The specification of `gradob` for FSQPD is as follows

```
      subroutine gradob(io,iprint,pd,nparam,j,rteps,udelta,x,fj,
     *                  gradfj,dummy)
      implicit double precision (a-h,o-z)
      dimension x(nparam),gradfj(nparam)
      logical pd
c
c     assign to gradfj the gradient of the jth objective function
c     evaluated at x
c
      return
      end
```

Arguments:

io	**(Input)** Output unit number.

`iprint`	**(Input)** Print level (see Section 3).
`pd`	**(Input)** Logical number.
`nparam`	**(Input)** Dimension of `x`.
`j`	**(Input)** Number of objective for which gradient is to be computed.
`rteps`	**(Input)** (used by `grobfd`) The square root of the machine precision `epsmac` (computed in FSQPD).
`udelta`	**(Input)** (used by `grobfd`) The mesh size the user wishes to use in computing the gradient of the objective function by finite difference.
`x`	**(Input)** Current iterate.
`fj`	**(Input)** (used by `grobfd`) Value of the objective at `x`.
`dummy`	**(Input)** Used by `grobfd`.
`gradfj`	**(Output)** Gradient of the `j`th objective function at x.

Note that `io`, `iprint`, `pd`, `rteps`, `udelta`, `fj`, and `dummy` are passed as arguments to `gradob` to allow for forward finite difference computation of the gradient.

5.4 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). The specification of `gradcn` for FSQPD is as follows

```
      subroutine gradcn (io,iprint,pd,nparam,j,rteps,udelta,x,gj,
     *                   gradgj,dummy)
      implicit double precision (a-h,o-z)
      dimension x(nparam),gradgj(nparam)
      external constr
      logical pd
c
c     assign to gradgj the gradient of the jth constraint
c     evaluated at x
c
      return
      end
```

Arguments:

`io`	**(Input)** Output unit number.
`iprint`	**(Input)** Print level (see Section 3).
`pd`	**(Input)** Logical number.
`nparam`	**(Input)** Dimension of `x`.
`j`	**(Input)** Number of constraint for which gradient is to be computed.
`rteps`	**(Input)** (used by `grcnfd`) The square root of the machine precision `epsmac` (computed by FSQPD).
`udelta`	**(Input)** (used by `grcnfd`) The mesh size the user wishes to use in computing the gradients of the constraints by finite difference.
`x`	**(Input)** Current iterate.
`gj`	**(Input)** (used by `grcnfd`) Value of the `j`th constraint at `x`.
`dummy`	**(Input)** Used by `grcnfd`.
`gradgj`	**(Output)** Gradient of the `j`the constraint evaluated at `x`.

Note that `io`, `iprint`, `pd`, `rteps`, `gj`, and `dummy` are passed as arguments to `gradcn` to allow for forward finite difference computation of the gradients.

6 Organization of FSQPD and Main Subroutines

6.1 Main Subroutines

FSQPD first checks for inconsistencies 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 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` $\geq$ `nnl`, etc.); and check that `eps` is 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 twice by FSQPD, first for generating a feasible iterate as explained at the end of Section 2 and second for generating an optimal iterate from that feasible iterate.

FSQPD1 uses the following subroutines:

`dir`	Compute search direction d_k and $\tilde{d}_k$.
`step`	Compute a step size along a certain search direction. It is also called to check if $x_k + d_k$ 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 the objective function by forward finite differences with mesh size equal to sign(x^i) $\times$ max{`udelta`, `rteps` $\times$ max(1, $\lvert x^i \rvert$)} 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 sign(x^i) $\times$ max{`udelta`, `rteps` $\times$ max(1, $\lvert x^i \rvert$)} for each component x^i of x (`rteps` is the square root of `epsmac`, the machine precision computed by FSQPD).

6.2 Other Subroutines

In addition to QPSOL and subroutines associated with it, the following subroutines are used:

`diagnl`	`dil`	`dqp`	`error`	`fuscmp`	`indexs`	`matrcp`
`matrvc`	`nullvc`	`qphess`	`scaprd`	`slope`	`small`	`subout`

7 Examples

The first problem is borrowed from [6] (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

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

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
      implicit double precision (a-h,o-z)
      parameter (iwsize=35)
      parameter (nwsize=275)
      dimension bl(3),bu(3),x(3),f(1),g(2),iw(iwsize),w(nwsize)
      external obj32,cntr32,grob32,grcn32
      logical Linfty
c
      nparam=3
      nf=1
      Linfty=.false.
      nnl=1
      nineq=1
      neq=1
c
      mode=0
      iprint=1
      miter=500
      bigbnd=1.d+10
      eps=1.d-08
      udelta=0.d0
c
      bl(1)=0.
      bl(2)=0.
      bl(3)=0.
      bu(1)=bigbnd
      bu(2)=bigbnd
      bu(3)=bigbnd
c
```

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

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

```
      function obj32(nparam,j,x)
      implicit double precision (a-h,o-z)
      dimension x(nparam)
c
      obj32=(x(1)+3.d0*x(2)+x(3))**2+4.d0*(x(1)-x(2))**2
      return
      end
c
      function cntr32(nparam,j,x)
      implicit double precision (a-h,o-z)
      dimension x(nparam)
c
      go to (10,20),j
 10   cntr32=x(1)**3-6.d0*x(2)-4.d0*x(3)+3.d0
      return
 20   cntr32=1.d0-x(1)-x(2)-x(3)
      return
      end
c
      subroutine grob32(io,iprint,pd,nparam,j,rteps,udelta,x,
     *                  fj,gradfj,dummy)
      implicit double precision (a-h,o-z)
      dimension x(nparam),gradfj(nparam)
      logical pd
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)=3.d0*fa-fb
      gradfj(3)=fa
      return
      end
c
      subroutine grcn32(io,iprint,pd,nparam,j,rteps,udelta,x,
     *                  gj,gradgj,dummy)
      implicit double precision (a-h,o-z)
      dimension x(nparam),gradgj(nparam)
      logical pd
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
```

After running the algorithm on a Sun 4/SPARCstation 1, the following output is obtained:

```
---------- FSQPD VERSION 2.0 : OUTPUT ------
The given initial point is feasible:
                    0.10000000000000e+00
                    0.70000000000000e+00
                    0.20000000000000e+00

iteration                              3
inform                                 0
x                   0.00000000000000e+00
                    0.00000000000000e+00
                    0.10000000000000e+01
objectives          0.10000000000000e+01
```

```
constraints       -0.10000000000000e+01
                   0.00000000000000e+00
SCV                0.00000000000000e+00
ktnorm             0.31401849173676e-15
ncallf                                3
ncallg                                6

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

Our second example is taken from example 6 in [7]. The problem is as follows.

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

where

$$f_i(x) = \tfrac{1}{15} + \tfrac{2}{15}(\textstyle\sum_{j=1}^{6} \cos(2\pi x_j \sin\theta_i) + \cos(7\pi \sin\theta_i)),$$
$$\theta_i = \tfrac{\pi}{180}(8.5 + 0.5i), \; i = 1, \ldots, 163,$$
$$s = 0.425.$$

The feasible initial guess is: $x_0 = (0.5, 1, 1.5, 2, 2, 5, 3)^T$ with $\max\limits_{i=1,\ldots,163} |f_i(x_0)| = 0.22051991555531$. A suitable main program is as follows.

```
c
c     problem description
c
      implicit double precision (a-h,o-z)
      parameter (iwsize=706)
      parameter (nwsize=9116)
      double precision x(6),bl(6),bu(6),iw(iwsize),w(nwsize),
     *                 f(163),g(7)
      external objmad,cnmad,grobfd,grcnfd
      logical Linfty
c
      mode=1
      iprint=1
      miter=500
      bigbnd=1.d+10
      eps=1.0d-10
      udelta=0.d0
c
      nparam=6
      nf=163
      Linfty=.true.
      nnl=0
      nineq=7
      neq=0
c
      do 10 i=1,6
        bl(i)=-bigbnd
        bu(i)=bigbnd
 10   continue
c
c     give the initial value of x
c
      x(1)=0.5
      x(2)=1.0
      x(3)=1.5
      x(4)=2.0
      x(5)=2.5
      x(6)=3.0
c
      call FSQPD(nparam,nf,Linfty,nnl,nineq,neq,mode,iprint,
     *           miter,inform,bigbnd,eps,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.

```
      function objmad(nparam,j,x)
      integer nparam,j,i
      double precision x(nparam),objmad,theta
```

```
c
      theta=3.14159d0*(8.5d0+j*0.5d0)/180.d0
      objmad=0.d0
      do 10 i=1,6
 10     objmad=objmad+dcos(2.d0*3.14159*x(i)*dsin(theta))
      objmad=2.d0*(objmad+cos(2.d0*3.14159*3.5*dsin(theta)))/15.d0+1.d0/15.d0
      return
      end
c
      function cnmad(nparam,j,x)
      integer nparam,j
      double precision x(nparam),cnmad,ss
c
      ss=0.425
      if(j.eq.1) then
        cnmad=ss-x(1)
        return
      endif
      if(j.gt.1.and.j.lt.7) cnmad=ss+x(j-1)-x(j)
      if(j.eq.7) cnmad=ss+x(6)-3.5
      return
      end
```

After running the algorithm on a Sun4/SPARCstation 1, the following output is obtained (the results for the set of objectives have been deleted to save space)

```
--------- FSQPD VERSION 2.0 : OUTPUT -------
The given initial point is feasible:
                   0.50000000000000E+00
                   0.10000000000000E+01
                   0.15000000000000E+01
                   0.20000000000000E+01
                   0.25000000000000E+01
                   0.30000000000000E+01

iteration                             8
x                  0.42500000000000E+00
                   0.85000000000000E+00
                   0.12750000000000E+01
```

```
                         0.17000000000000E+01
                         0.21840758252977E+01
                         0.28732752473301E+01
    objective max4       0.11421845223065E+00
    objmax               0.11310472703986E+00
    constraints          0.00000000000000E+00
                         0.00000000000000E+00
                         0.00000000000000E+00
                         0.00000000000000E+00
                        -0.59075825297727E-01
                        -0.26419942203233E+00
                        -0.20172475266995E+00
    SCV                  0.00000000000000E+00
    ktnorm               0.93448637498563E-15
    ncallf                               1304
    ncallg                                  0

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

8 Results for Test Problems

These results are provided to allow the user to compare FSQP with his/her favorite code (see also [2]). Table 1 contains results obtained for some (non-minimax) test problems from [6] (the same initial points as in [6] were selected). `prob` indicates the problem number as in [6], `nnl` 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 user-provided Kuhn-Tucker norm requirement, `SCV` the sum of feasibility violation of linear constraints (see section 4). On each test problem, `eps` was selected so as to achieve the same field precision as in [6]. The value of parameter `mode` (0 for FSQP-AL, 1 for FSQP-NL) is indicated in column "mode". All these results were obtained on a Sun 4/SPARCstation 1.

Results obtained on selected (possibly constrained) minimax problems are summarized in Table 2. Problems `bard`, `davd2`, `f&r`, `hettich`, and `wats` are from [8]; `cb2`, `cb3`, `r-s`, `wong` and `colv` are from [9; Examples 5.1-5] (the latest test results on problems `bard` down to `wong` can be found in [10]); `kiw1` and `kiw4` are from [11] (results for `kiw2` and `kiw3` are not reported due to data disparity); `mad1` to `mad8` are from [7, Examples 1-8]; `polk1` to `polk4` are from [12]. 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 [6] by removing certain constraints and including instead additional objectives of the form

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

where the α_i's are positive scalars and $g_j(x) \leq 0$. 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 mode, nnl, 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 wong, execution was terminated when $\|d_k^0\|$ is smaller than the corresponding value of ε in the column of eps (this was also done for problems p43m through p117m); for problems kiw1 and kiw4, execution was terminated when the 14 significant digits carried out by the output of our codes did not change and the values in the column of eps are $\|d_k^0\|$ at the stopping point; 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; for problems polk1 through polk4, execution was terminated when $\log_e \|x_k - x^*\|$ is smaller than the corresponding value of ε in the column of eps.

Acknowledgment

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

prob	mode	nnl	ncallf	ncallg	iter	objective	ktnorm	eps	SCV
p12	0	1	7	15	7	−.300000000E+02	.72E-06	.10E-05	.0
	1		7	13	7	−.300000000E+02	.79E-06	.10E-05	.0
p29	0	1	12	23	11	−.226274170E+02	.13E-07	.10E-05	.0
	1		13	17	13	−.226274170E+02	.19E-06	.10E-05	.0
p30	0	1	16	31	16	.100000000E+01	.54E-08	.10E-07	.0
	1		15	15	15	.100000000E+01	.97E-08	.10E-07	.0
p31	0	1	9	21	8	.600000000E+01	.23E-05	.10E-04	.0
	1		10	19	10	.600000000E+01	.46E-06	.10E-04	.0
p32	0	1	3	6	3	.100000000E+01	.31E-15	.10E-07	.0
	1		3	4	3	.100000000E+01	.31E-15	.10E-07	.0
p33	0	2	4	14	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-09	.10E-07	.0
p43	0	3	11	62	9	−.440000000E+02	.12E-05	.10E-04	.0
	1		13	55	13	−.440000000E+02	.86E-06	.10E-04	.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	7	9	3	.306463061E−01	.29E-05	.10E-04	.0
	1		7	8	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	.63E-11	.10E-07	.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	42	4	−.528033513E+07	.68E-12	.10E-08	.0
	1		4	30	4	−.528033513E+07	.66E-09	.10E-08	.0
p86	0	0	14	0	9	−.323486790E+02	.17E-13	.10E-07	.0
	1		8		7	−.323486790E+02	.17E-13	.10E-07	.0
p93	0	2	15	61	12	.135075968E+03	.37E-03	.10E-02	.0
	1		15	38	15	.135075964E+03	.41E-04	.10E-02	.0
p100	0	4	23	168	16	.680630057E+03	.62E-06	.10E-03	.0
	1		20	128	17	.680630057E+03	.26E-04	.10E-03	.0
p110	0	0	10	0	9	−.457784697E+02	.86E-10	.10E-07	.0
	1		10		9	−.457784697E+02	.86E-10	.10E-07	.0
p113	0	5	12	122	12	.243063768E+02	.81E-03	.10E-02	.0
	1		12	106	12	.243064357E+02	.85E-03	.10E-02	.35E-14
p117	0	5	20	219	19	.323486790E+02	.58E-04	.10E-03	.0
	1		18	94	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	.13E-14	.10E-07	.0

Table 1: Results for General Problems

prob	mode	nnl	nf	ncallf	ncallg	iter	objmax	ktnorm	eps	SCV
bard	0	0	15	225	0	8	.508163265E−01	.11E-12	.50E-05	.0
	1			105		7	.508168686E−01	.22E-06	.50E-05	.0
cb2	0	0	3	33	0	6	.195222453E+01	.37E-06	.50E-05	.0
	1			18		6	.195222453E+01	.30E-05	.50E-05	.0
cb3	0	0	3	15	0	3	.200000000E+01	.40E-05	.50E-05	.0
	1			15		5	.200000000E+01	.47E-08	.50E-05	.0
colv	0	0	6	181	0	15	.274053332E+02	.72E-04	.50E-05	.0
	1			84		14	.274053332E+02	.45E-05	.50E-05	.0
davd2	0	0	20	380	0	10	.115706440E+03	.12E-05	.50E-05	.0
	1			220		10	.115706440E+03	.11E-05	.50E-05	.0
f&r	0	0	2	34	0	9	.494895210E+01	.17E-06	.50E-05	.0
	1			20		10	.494895210E+01	.14E-06	.50E-05	.0
hettich	0	0	5	96	0	10	.245935695E−02	.68E-07	.50E-05	.0
	1			55		10	.245939485E−02	.63E-07	.50E-05	.0
r-s	0	0	4	75	0	9	−.440000000E+02	.10E-04	.50E-05	.0
	1			53		10	−.440000000E+02	.81E-05	.50E-05	.0
wats-6	0	0	31	713	0	12	.127170954E−01	.36E-07	.50E-05	.0
	1			434		13	.127170913E−01	.10E-09	.50E-05	.0
wats-20	0	0	31	2757	0	42	.138908355E−07	.14E-06	.50E-05	.0
	1			1395		43	.141191856E−07	.12E-06	.50E-05	.0
wong	0	0	5	232	0	20	.680630057E+03	.24E-04	.50E-05	.0
	1			191		26	.680630057E+03	.14E-03	.50E-05	.0
kiw1	0	0	10	213	0	10	.226001621E+02	.10E-04	.11E-05	.0
	1			140		13	.226001621E+02	.54E-05	.60E-06	.0
kiw4	0	0	2	38	0	8	.222004460E−15	.42E-07	.42E-07	.0
	1			28		10	.222004460E−15	.21E-07	.15E-07	.0
mad1	0	0	3	27	0	5	−.389659516E+00	.25E-15	.10E-11	.0
	1			18		6	−.389659516E+00	.48E-10	.10E-11	.0
mad2	0	0	3	63	0	11	−.330357143E+00	.75E-09	.10E-11	.0
	1			58		18	−.330357143E+00	.23E-08	.10E-11	.0
mad4	0	0	3	33	0	6	−.448910786E+00	.44E-15	.10E-11	.0
	1			24		8	−.448910786E+00	.44E-15	.10E-11	.0
mad5	0	0	3	39	0	7	−.100000000E+01	.16E-15	.10E-11	.0
	1			24		8	−.100000000E+01	.79E-14	.10E-11	.0
mad6	0	0	163	1793	0	6	.113104635E+00	.81E-11	.10E-11	.0
	1			1304		8	.113104727E+00	.93E-15	.10E-11	.0
mad8-10	0	0	18	342	0	10	.381173963E+00	.41E-12	.10E-11	.0
	1			252		14	.381173963E+00	.46E-15	.10E-11	.0
mad8-30	0	0	58	1740	0	15	.547620496E+00	.70E-15	.10E-11	.0
	1			1160		18	.547620496E+00	.48E-15	.10E-11	.0
mad8-50	0	0	98	3822	0	20	.579276202E+00	.98E-15	.10E-11	.0
	1			2058		21	.579276202E+00	.82E-13	.10E-11	.0
polk1	0	0	2	42	0	10	.271828183E+01	.50E-04	−10.00	.0
	1			22		10	.271828183E+01	.62E-04	−10.00	.0
polk2	0	0	2	187	0	36	.545981500E+02	.52E-03	− 9.00	.0
	1			143		53	.545981500E+02	.75E-06	− 9.00	.0
polk3	0	0	10	253	0	12	.370348302E+01	.23E-02	− 5.50	.0
	1			141		12	.370348451E+01	.26E-02	− 5.50	.0
polk4	0	0	3	45	0	7	.233517650E−12	.84E-05	−10.00	.0
	1			24		7	.364606056E+00	.26E-04	−10.00	.0
p43m	0	1	3	81	36	14	−.440000000E+02	.30E-06	.50E-05	.0
	1			60	25	16	−.440000000E+02	.39E-05	.50E-05	.0
p84m	0	4	3	21	28	4	−.528033513E+07	.0	.50E-05	.0
	1			9	12	3	−.528033513E+07	.37E-03	.50E-05	.0
p113m	0	5	4	100	142	13	.243062091E+02	.31E-05	.50E-05	.0
	1			84	115	15	.243062091E+02	.31E-05	.50E-05	.0
p117m	0	3	3	144	124	21	.323486790E+02	.46E-05	.50E-05	.0
	1			58	54	17	.323486790E+02	.26E-04	.50E-05	.0

Table 2: Results for Minimax Problems

References

[1] E.R. Panier & A.L. Tits, "On Feasibility, Descent and Superlinear Convergence in Inequality Constrained Optimization," Systems Research Center, University of Maryland, Technical Report SRC-TR-89-27, College Park, MD 20742, 1989.

[2] J.F. Bonnans, E.R. Panier, A.L. Tits & J. Zhou, "Avoiding the Maratos Effect by Means of a Nonmonotone Line Search. II. Inequality Constrained Problems – Feasible Iterates," Systems Research Center, University of Maryland, Technical Report SRC-TR-89-42r1, College Park, MD 20742, 1989.

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

[4] 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.

[5] 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.

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

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

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

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

[10] 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 .

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

[12] 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).