ISR

User's Guide for CFSQP Version 2.0: A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality Constraints[1]

Craig Lawrence, Jian L. Zhou, and André L. Tits

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

(Institute for Systems Research TR-94-16)

[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.

Abstract

CFSQP is a set of C functions 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, CFSQP first generates a feasible point for these constraints; subsequently the successive iterates generated by CFSQP 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. When solving problems with many sequentially related constraints (or objectives), such as discretized semi-infinite programming (SIP) problems, CFSQP gives the user the option to use an algorithm that efficiently solves these problems, greatly reducing computational effort. The user has the option of either requiring that the objective function (penalty function if nonlinear equality constraints are present) 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 functions that define the objective functions and constraint functions and may either provide functions to compute the respective gradients or require that CFSQP estimate them by forward finite differences.

CFSQP is an implementation of 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 constraints, or an exact penalty function if nonlinear equality constraints are present.

Conditions for External Use

1. The CFSQP 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 CFSQP routines in research reports or publications. Whenever such reports are released for public access, a copy should be forwarded to the authors.
4. The CFSQP routines may only by used for research and development, unless it has been agreed otherwise with the authors in writing.

User's Guide for CFSQP Version 2.0 (Released February 1994)

Enquiries should be directed to

Prof. André L. Tits
Electrical Engineering Dept.
and Institute for Systems Research
University of Maryland
College Park, Md 20742
U. S. A.
Phone : 301-405-3669
Fax : 301-405-6707
E-mail : andre@eng.umd.edu

Contents

1 Introduction

CFSQP (C code for Feasible Sequential Quadratic Programming) is a set of C functions 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. In addition, CFSQP contains special provisions for efficiently handling problems with many sequentially related objectives/constraints, for example discretized Semi-Infinite Programming (SIP) problems.

In the case when no sequentially related constraints or objectives are present, CFSQP tackles optimization problems of the form

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

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

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

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

In full generality, i.e. including sequentially related objectives and constraints, CFSQP handles problems of the form

$$(P_{sr}) \qquad \text{minimize } \max\{\max_{i \in I^f} f_i(x), \max_{i \in I^{sr}} \max_{\omega \in \Omega^{f_i}} f_i(x, \omega)\} \quad \text{s.t. } \; x \in X$$

where $I^f = \{1, \ldots, n_f - n_{fsr}\}$, $I^{sr} = \{n_f - n_{fsr} + 1, \ldots, n_f\}$ ($I^{sr} = \emptyset$ if $n_{fsr} = 0$), Ω^{f_i} is an index set for objective functions that are somehow sequentially related, and X is now the set of points $x \in \mathbf{R}^n$ satisfying

$$\begin{array}{l} bl \leq x \leq bu \\ g_j(x) \leq 0, \quad j = 1, \ldots, n_i - n_{sr} \\ g_j(x, \xi) \leq 0, \quad \forall \xi \in \Xi^{g_j}, \quad j = n_i - n_{sr} + 1, \ldots, n_i \\ g_j(x) \equiv \langle c_{j-n_i}, x \rangle - d_{j-n_i} \leq 0, \quad j = n_i + 1, \ldots, t_i - \ell_{sr} \\ g_j(x, \xi) \equiv \langle c_{j-n_i}(\xi), x \rangle - d_{j-n_i}(\xi) \leq 0, \quad \forall \xi \in \Xi^{g_j}, \quad j = t_i - \ell_{sr} + 1, \ldots, t_i \\ h_j(x) = 0, \quad j = 1, \ldots, n_e \\ h_j(x) \equiv \langle a_{j-n_e}, x \rangle - b_{j-n_e} = 0, \quad j = n_e + 1, \ldots, t_e \end{array}$$

with $bl \in \mathbf{R}^n$; $bu \in \mathbf{R}^n$; $f_i : \mathbf{R}^n \to \mathbf{R}$, $i \in I^f$ smooth; $f_i : \mathbf{R}^n \times \Omega^{f_i} \to \mathbf{R}$, $i \in I^{sr}$ continuously differentiable with respect to the first argument for each $\omega \in \Omega^{f_i}$; $g_j : \mathbf{R}^n \to \mathbf{R}$, $j = 1, \ldots, n_i - n_{sr}$ nonlinear and smooth; $g_j : \mathbf{R}^n \times \Xi^{g_j} \to \mathbf{R}$, $j = n_i - n_{sr} + 1, \ldots, n_i$ nonlinear, continuously differentiable with respect to the first argument for each $\xi \in \Xi^{g_j}$; $c_{j-n_i} \in \mathbf{R}^n$, $d_{j-n_i} \in \mathbf{R}$, $j = n_i + 1, \ldots, t_i - \ell_{sr}$; $c_{j-n_i} : \Xi^{g_j} \to \mathbf{R}^n$, $d_{j-n_i} : \Xi^{g_j} \to \mathbf{R}$, $j = t_i - \ell_{sr} + 1, \ldots, t_i$; $h_j : \mathbf{R}^n \to \mathbf{R}$, $j = 1, \ldots, n_e$ nonlinear and smooth; $a_j \in \mathbf{R}^n$, $b_j \in \mathbf{R}$, $j = 1, \ldots, t_e - n_e$.

From this point forward, in order to ease the presentation of the algorithm, we discuss problem (P), postponing discussion of the algorithm used to solve (P_{sr}) until § 3. If the initial guess provided by the user is infeasible for linear constraints, CFSQP generates a point satisfying these constraints by solving a strictly convex quadratic program (QP). Next, if the initial guess, or the newly generated initial guess, is infeasible for the nonlinear inequality constraints, CFSQP generates a point x_0 satisfying all constraints (other than nonlinear equality constraints) by iterating on the problem of minimizing the maximum of the nonlinear inequality constraints. Then, using a scheme due to Mayne and Polak [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 that are iteratively adjusted. The resulting optimization problem therefore involves only linear constraints and nonlinear inequality constraints. The successive iterates generated by CFSQP 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 no nonlinear equality constraints are present) decrease at each iteration (after feasibility for original nonlinear inequality and linear constraints has been reached), or requiring a decrease within at most four iterations. He/She must provide functions that define the objectives and constraints, and may either provide functions to compute the respective gradients or require that CFSQP estimate them by forward finite differences.

Thus, CFSQP 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], [4], [5] and [6], 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

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

is used (along the arc described below for the monotone line search) when minimizing the maximum of the nonlinear inequality constraints to generate an initial feasible point. 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 [7] and analyzed in [3] and [4] in the present 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" to yield a feasible direction if nonlinear constraints are present, 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.

When sets of many sequentially related objectives or constraints are present (e.g., when the problem at hand involves finely discretized semi-infinite objectives or constraints), the user may request that, at each iteration, CFSQP select a small subset of these objectives/constraints for inclusion in the quadratic programming subproblems, thus possibly saving considerable time and computational effort. The method of selecting appropriate subsets of the sequentially related constraints and objectives is outlined and analyzed in [5] and [6].

For the solution of the quadratic programming subproblems, CFSQP 2.0 is set up to call a C version of QLD [8], converted from Fortran via `f2c` (see [9]) and provided with the CFSQP distribution for the user's convenience.

2 Description of the Basic Algorithms

The algorithms described and analyzed in [2], [3], and [4] are as follows. For simplicity of exposition, we describe the algorithms as they pertain to solving problem (P), deferring the discussion of the algorithm described in [5] and [6] for the solution of (P_{sr}) until § 3. 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 feasible direction $d^1 = d^1(x)$ is computed, which is nonzero even at Karush-Kuhn-Tucker points (and thus is not everywhere a descent direction). 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 for d^1. As in [3], $\tilde{d}$ is required to initialize superlinear convergence.

The CFSQP 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. 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, CFSQP 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 following notation:

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

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

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

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

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 the first 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. $x_0 \in \mathbf{R}^n$, $\epsilon > 0$, $\epsilon_e > 0$ 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

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

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

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

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 $n_i+n_e\ne0$ and $\delta_k=-\langle d_k^0,H_kd_k^0\rangle$ otherwise. Compute t_k, the first number t in the sequence $\{1,\beta,\beta^2,\dots\}$ satisfying

$$\begin{gathered} f_m(x_k+td_k+t^2\tilde{d}_k,p_k)\le f_m(x_k,p_k)+\alpha t\delta_k \\ g_j(x_k+td_k+t^2\tilde{d}_k)\le0,\quad j=1,\dots,n_i \\ \langle c_{j-n_i},x_k+td_k+t^2\tilde{d}_k\rangle\le d_{j-n_i},\quad \forall j>n_i\ \&\ j\notin I_k^g(d_k) \\ h_j(x_k+td_k+t^2\tilde{d}_k)\le0,\quad j=1,\dots,n_e. \end{gathered}$$

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 $t < \bar{t}_k$, it may be more efficient to first check the function that caused the previous trial value of t to be rejected (intuitively, it may be more likely to fail the test again). If this function is a constraint, it will always be checked first, followed by all other constraints, then objectives. If it is an objective however, there are two alternatives available to the user (see `mode` in § 4): (i) the "problem" objective is checked first, followed by all other objectives, then constraints; or (ii) all constraints are checked first, followed by the "problem" objective, then the other objectives. The former is likely more effective; the latter is helpful when some objectives are not defined outside the feasible set.

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 [10].

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

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

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

  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*.

□

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

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

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.

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

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

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

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

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

and

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

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

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

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 $n_i + n_e \neq 0$, 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

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

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

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

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

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 $t \leq \bar{t}_k$. 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 $t < \bar{t}_k$, either the function that caused the previous value of t to be rejected is checked first and all functions of the same type ("objective" or "constraint") as the latter will then be checked first; or constraints will be always checked first (if it is a constraint that caused the previous value of t to be rejected, that constraint will be checked first; see `mode` in § 4).

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 [10].

- 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, if $\rho_k^\ell < 1$, set $C_{k+1} = 10C_k$.

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

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

  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.[4]

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

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

- Increase k by 1.

- Go back to *Step 1*.

□

Remark: The Hessian matrix is reset in both algorithms whenever stepsize is very small and the updating of the matrix has gone through $5n$ iterations. This is helpful in some situations where the Hessian matrix becomes singular.

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

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

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

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

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

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

3 Refinements for the Case of Many Objectives/Constraints

As mentioned in the introduction, CFSQP is equipped to handle in an efficient manner problems involving many sequentially related objectives or constraints, e.g., finely discretized problems from Semi-Infinite Programming (SIP). The algorithm employed is described and analyzed in [5] and [6]. Below we describe the algorithm as implemented in CFSQP, omitting "isolated" objectives and constraints for simplicity of exposition. The essential difference to the algorithm in the previous section is that at iteration k only subsets $\Omega_k^{f_i} \subset \Omega^{f_i}$ and $\Xi_k^{g_i} \subset \Xi^{g_i}$ of the sets of sequentially related objectives and constraints are considered when solving the quadratic programming subproblems used to construct the search direction, possibly saving considerable time and computational effort. The "active" sets are updated in such a way that global and fast local convergence is still assured.

In order to further simplify the exposition we consider a problem with one "set" of sequentially related constraints and one "set" of sequentially related objectives (CFSQP can handle problems with multiple such sets, as well as many isolated objectives and constraints). For example, the problem presented below could correspond to a discretized semi-infinite program with one functional objective and one functional constraint. To ease notation, we let $\Omega := \Omega^{f_1}$ and $\Xi := \Xi^{g_1}$. Hence, the problem we will address is

$$\begin{array}{ll} \min\limits_{x\in\mathbf{R}^n} & \max\limits_{\omega\in\Omega} f(x,\omega) \\ \text{s.t.} & g(x,\xi) \le 0 \quad \forall \xi \in \Xi. \end{array}$$

We present here the algorithm corresponding to FSQP-AL, *i.e.* the Armijo-type line search. The nonmonotone line search (algorithm FSQP-NL) is also available for these problems, and the corresponding algorithm is a parallel modification of the algorithm FSQP-NL presented in § 2. Given $x \in \mathbf{R}^n$, let

$$\Phi(x) = \max_{\omega \in \Omega} f(x,\omega).$$

Additionally, given $\hat{\Omega} \subset \Omega$, define

$$\Phi_{\hat{\Omega}}(x) = \max_{\omega \in \hat{\Omega}} f(x,\omega).$$

Further, given a direction $d \in \mathbf{R}^n$, let

$$\Phi'_{\hat{\Omega}}(x,d) = \max_{\omega \in \hat{\Omega}} \{f(x,\omega) + \langle \nabla_x f(x,\omega), d\rangle\} - \Phi_{\hat{\Omega}}(x),$$

which is a first order approximation to $\Phi_{\hat{\Omega}}(x+d) - \Phi_{\hat{\Omega}}(x)$. Finally, given $\tilde{d} \in \mathbf{R}^n$ define

$$\tilde{\Phi}'_{\hat{\Omega}}(x+d,x,\tilde{d}) = \max_{\omega \in \hat{\Omega}} \{f(x+d,\omega) + \langle \nabla_x f(x,\omega), \tilde{d}\rangle\} - \Phi_{\hat{\Omega}}(x+d).$$

At each iteration k, subsets $\Omega_k \subset \Omega$ and $\Xi_k \subset \Xi$ will be used to compute the search direction. Given $x \in \mathbf{R}^n$ and letting $\zeta_{k,\omega}$ denote the multiplier associated with the particular objective defined by ω at the kth iteration, and letting $\lambda_{k,\xi}$ denote the multiplier associated with the particular constraint defined by ξ at the kth iteration, we define the following sets to be included in Ω_k and Ξ_k:

$$\Omega_{\max}(x) = \{\omega \in \Omega : f(x,\omega) = \Phi(x)\}$$

$$\Xi_{\text{act}}(x) = \{\xi \in \Xi : g(x,\xi) = 0\},$$

and the "binding" objectives and constraints from the previous iteration:

$$\Omega_k^b = \{\omega \in \Omega_k : \zeta_{k,\omega} > 0\}$$

$$\Xi_k^b = \{\xi \in \Xi_k : \lambda_{k,\xi} > 0\}.$$

In addition, some heuristics are used to increase the number of constraints and objectives in Ω_k and Ξ_k, with the hopes that performance will be improved while still maintaining lower computational effort. Specifically, for some $\epsilon > 0$, we consider the "ϵ-active left local maximizers" at x_k, which, for the objectives, we shall denote Ω_ϵ^{llm}. Using the notation and definitions from [5], a discretization point $\omega_i \in \Omega := \{\omega_1, \ldots, \omega_{|\Omega|}\}$ is called ϵ-active if it is in the set

$$\Omega_\epsilon(x) = \{\omega_i \in \Omega : f(x,\omega_i) > \Phi(x) - \epsilon\}.$$

We call the point a left-local maximizer if it satisfies *one* of the following three conditions: (i) $i \in \{2, \ldots, |\Omega| - 1\}$ and

$$f(x, \omega_i) > f(x, \omega_{i-1}) \tag{1}$$

and

$$f(x, \omega_i) \geq f(x, \omega_{i+1}); \tag{2}$$

(ii) $i = 1$ and (2); (iii) $i = |\Omega|$ and (1). The equivalent set for constraints, Ξ_ϵ^{llm} is defined in an analogous way, except that the $\Phi(x)$ in the definition of $\Omega_\epsilon(x)$ should be replaced with a 0 in the definition of $\Xi_\epsilon(x)$.

At each iteration k, the SQP direction d_k^0 is computed as the solution of the quadratic program $QP_0(x_k, H_k, \Omega_k, \Xi_k)$, defined at x_k for H_k symmetric and positive definite by

$$(QP_0(x_k, H_k, \Omega_k, \Xi_k)) \qquad \begin{array}{cl} \min\limits_{d^0 \in \mathbf{R}^n} & \frac{1}{2}\langle d^0, H_k d^0 \rangle + \Phi'_{\Omega_k}(x_k, d^0) \\ \text{s.t.} & g(x_k, \xi) + \langle \nabla_x g(x_k, \xi), d^0 \rangle \leq 0, \quad \forall \xi \in \Xi_k. \end{array}$$

Finally, we need to slightly modify the definitions of $I_k^f(d_k)$ and $I_k^g(d_k)$ (used in the computation of $\tilde{d}$) from those presented in the last section. Specifically, define the set of active objective functions

$$I_k^f(d_k) = \{\omega \in \Omega : |f(x_k, \omega) - f(x_k, \hat{\omega})| \leq 0.2\|d_k\| \cdot \|\nabla_x f(x_k, \omega) - \nabla_x f(x_k, \hat{\omega})\|\},$$

where $\hat{\omega}$ is the first element of Ω such that $\zeta_{k,\hat{\omega}} > 0$. The set of "active" constraints is re-defined as

$$I_k^g(d_k) = \{\xi \in \Xi : |g(x_k, \xi)| \leq 0.2\|d_k\| \cdot \|\nabla_x g(x_k, \xi)\|\}.$$

We are now ready to present the simplified algorithm.

Algorithm FSQP-SR.

Parameters. $\eta = 0.1$, $\nu = 0.01$, $\alpha = 0.1$, $\beta = 0.5$, $\kappa = 2.1$, $\tau_1 = \tau_2 = 2.5$, $\underline{t} = 0.1$, $\epsilon_1 = 1$, $\epsilon_s = 1$, $\delta = 5$, $0 < \delta_s \ll 1$.

Data. $x_0 \in \mathbf{R}^n$, $\epsilon > 0$, $\epsilon_e > 0$.

Step 0: Initialization. Set $k = 0$, $H_0 =$ the identity matrix and $nset = 0$. If x_0 is infeasible for some constraint other than a nonlinear equality constraint, substitute a feasible point, obtained as described at the end of § 2. Set $\Omega_0 = \Omega_{\max}(x_0) \cup \Omega_\epsilon^{llm}(x_0) \cup \{\omega_1\} \cup \{\omega_{|\Omega|}\}$ and $\Xi_0 = \Xi_{\text{act}}(x_0) \cup \Xi_\epsilon^{llm}(x_0) \cup \{\xi_1\} \cup \{\xi_{|\Xi|}\}$.

Step 1: Computation of a search arc.

i. Compute d_k^0, the solution of the quadratic program $QP_0(x_k, H_k, \Omega_k, \Xi_k)$. If $\|d_k^0\| \leq \epsilon$ stop. If $\Xi = \emptyset$ set $d_k = d_k^0$ and go to *Step 1 iv.*

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

$$\begin{array}{ll} \displaystyle\min_{d^1 \in \mathbf{R}^n, \gamma \in \mathbf{R}} & \frac{\eta}{2}\langle d_k^0 - d^1, d_k^0 - d^1 \rangle + \gamma \\ \text{s.t.} & \Phi'_{\Omega_k}(x_k, d^1) \leq \gamma \\ & g(x_k, \xi) + \langle \nabla_x g(x_k, \xi), d^1 \rangle \leq \gamma, \quad \forall \xi \in \Xi_k \end{array}$$

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

$$\begin{array}{ll} \displaystyle\min_{\tilde{d} \in \mathbf{R}^n} & \frac{1}{2}\langle (d_k + \tilde{d}), H_k(d_k + \tilde{d}) \rangle + \tilde{\Phi}'_{I_k^f(d_k) \cup \Omega_k}(x_k, d_k, \tilde{d}) \\ \text{s.t.} & g(x_k + d_k, \xi) + \langle \nabla_x g(x_k, \xi), \tilde{d} \rangle \leq -\min(\nu\|d_k\|, \ \|d_k\|^{\tau_2}), \quad \forall \xi \in I_k^g(d_k) \cup \Xi_k \end{array}$$

If the quadratic program has no solution or if $\|\tilde{d}_k\| > \|d_k\|$, set $\tilde{d}_k = 0$.

Step 2. Arc search. Let $\delta_k = \Phi'_\Omega(x_k, d_k)$ if $\Xi \neq \emptyset$ and the constraints are nonlinear in x. Let $\delta_k = -\langle d_k^0, H_k d_k^0 \rangle$ otherwise. Compute t_k, the first number t in the sequence $\{1, \beta, \beta^2, \ldots\}$ satisfying

$$\begin{array}{l} \Phi(x_k + td_k + t^2\tilde{d}_k) \leq \Phi(x_k) + \alpha t \delta_k \\ g(x_k + td_k + t^2\tilde{d}_k, \xi) \leq 0, \quad \forall \xi \in \Xi. \end{array}$$

The specifics of the line search are precisely the same as given in § 2, and will not be repeated here.

Step 3. Updates.

- Set

$$\Omega_{k+1} = \Omega_{\max}(x_{k+1}) \cup \Omega_k^b \cup \Omega_\epsilon^{llm}(x_{k+1})$$

and

$$\Xi_{k+1} = \Xi_{\text{act}}(x_{k+1}) \cup \Xi_k^b \cup \Xi_\epsilon^{llm}(x_{k+1})$$

If $t_k < 1$ and the last stepsize reduction was due to a sequentially related objective with index $\bar{\omega}$, then set

$$\Omega_{k+1} = \Omega_{k+1} \cup \{\bar{\omega}\}.$$

If $t_k < 1$ and the last stepsize reduction was due to a sequentially related constraint with index $\bar{\xi}$, then set

$$\Xi_{k+1} = \Xi_{k+1} \cup \{\bar{\xi}\}.$$

- If $nset > 5n$ and $t_k < \underline{t}$, set $H_{k+1} = H_0$ and $nset = 0$. If $t_k \leq \delta_s$ and the discretization point causing a violation during the line search was *not* in Ω_k or Ξ_k, set $H_{k+1} = H_k$. 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 [10].
- 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*.

$\square$

In the case where there is more than one set of sequentially related objectives or constraints, the above algorithm is modified only slightly. In particular, for *each* objective set we compute $\Omega^{f_i}_{\max}(x)$, Ω_k^{b,f_i}, etc., and each set $\Omega_k^{f_i}$ is constructed as Ω_k in the above algorithm (i.e., now there is an "Ω_k" for each objective set). Likewise for multiple constraint sets. The definitions of the various $\Phi(\cdot)$ functions have to be modified accordingly, as well. The main idea is that once we have determined the subsets of constraints and objectives to use in the computation of the search directions, the algorithm is the same whether we originally had one set, or many sets.

4 Specification of CFSQP

The specification of CFSQP is as follows (an ANSI compliant definition is automatically used on compilers that expect such a definition):

```
void
cfsqp(nparam,nf,nfsr,nineqn,nineq,neqn,neq,ncsrl,ncsrn,mesh_pts,
      mode,iprint,miter,inform,bigbnd,eps,epseqn,udelta,bl,bu,x,
      f,g,lambda,obj,constr,gradob,gradcn)
int     nparam,nf,nfsr,neqn,nineqn,nineq,neq,ncsrl,ncsrn,mode,
        iprint,miter,*mesh_pts,*inform;
double  bigbnd,eps,epseqn,udelta;
double  *bl,*bu,*x,*f,*g,*lambda;
void    (* obj)(),(* constr)(),(* gradob)(),(* gradcn)();
```

Important: all real variables (arrays) must be declared as double precision (pointers to double precision arrays) in the routine that calls CFSQP.

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

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

nfsr **(Input)** Number of sets of sequentially related objective functions (n_{fsr} in the problem 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).

ncsrl **(Input)** Number (possibly zero) of sets of linear sequentially related constraints (ℓ_{sr} in the problem description).

ncsrn **(Input)** Number (possibly zero) of sets of nonlinear sequentially related constraints (n_{sr} in the problem description).

mesh_pts **(Input)** Pointer to an array of integers of length $n_{fsr} + n_{sr} + \ell_{sr}$ indicating the number of objectives/constraints in each specific set of sequentially related objectives/constraints ($|\Omega^{f_i}|$ and $|\Xi^{g_j}|$ in the problem description). Elements $0, \ldots,$ `nfsr` $- 1$ should contain the number of objectives in each sequentially related objective set, the next `ncsrn` elements should contain the number of constraints in each sequentially related nonlinear constraint set, and the final `ncsrl` elements should contain the number of constraints in each sequentially related linear constraint set.

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

`A` $= 0$: (P) is to be solved.

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

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

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 $i \in I^f$. If sequentially related objectives are present, equivalent modifications are made to (P_{ST}).

$\mathtt{B} = 0$: Algorithm FSQP-AL is selected, resulting in a decrease of the (modified) objective function at each iteration.

$\mathtt{B} = 1$: 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).

$\mathtt{C} = 1$: For $t < \bar{t}_k$ (see the end of algorithm statement) during the line search, the function that caused the previous value of t to be rejected is checked first and all functions of the same type ("objective" or "constraint") as the latter will then be checked first. (Recommended for most users.)

$\mathtt{C} = 2$: Constraints will be always checked first at each trial point during the line search. If it is a constraint that caused the previous value of t to be rejected, that constraint will be checked first. (Useful when objective functions are not defined or are difficult to evaluate outside of the feasible region; not however that if gradients are evaluated by finite differences, in rare instances, objectives functions may be evaluated at infeasible "perturbed" points).

iprint **(Input)** Parameter indicating the desired output (see § 6 for details):

`iprint` $= 0$: No information other than user-input errors is displayed. This value is imposed during phase 1.

`iprint` $= 1$: Objective and constraint values at the initial feasible point are displayed. 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, sum of feasibility violation, and if appropriate, the total number of individual constraints/objectives used from the sets of sequentially related constraints/objectives used during the final iteration 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.

`iprint` = $10{*}N + M$: N any positive integer, M=2 or 3. Information corresponding to `iprint`=M is displayed at every $(10 \times N)$th iteration and at the last iteration.

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

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

`inform` = 0 : Normal termination of execution in the sense that either $\|d^0\| \leq$ `eps` and (if `neqn` $\neq 0$) $\sum_{j=1}^{n_e} |h_j(x)| \leq$ `epseqn` or one of the user-supplied stopping criteria is satisfied (see § 5).

`inform` = 1 : The user-provided initial guess is infeasible for linear constraints and CFSQP is unable to generate a point satisfying all these constraints.

`inform` = 2 : The user-provided initial guess is infeasible for nonlinear inequality constraints and linear constraints; and CFSQP is unable to generate a point satisfying all these constraints. This may be due to insufficient accuracy of the QP solver.

`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 CFSQP).

`inform` = 5 : Failure of the QP solver in attempting to construct d^0. A more robust QP solver may succeed.

`inform` = 6 : Failure of the QP solver in attempting to construct d^1. A more robust QP solver may succeed.

`inform` = 7 : Input data are not consistent (with printout indicating the error).

`inform` = 8 : New iterate is numerically equivalent to the previous iterate, though the stopping criterion is not yet satisfied. Relaxing the stopping criterion should solve this problem.

bigbnd **(Input)** (see also `bl` and `bu` below) Plays the role of an "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 CFSQP). (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 track of the process of optimization and terminate CFSQP at an 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 CFSQP).

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 `0.e0` 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., `bl[j]` $= -\infty$ for some j), the value used must satisfy `bl[j]` $\leq$ `-bigbnd`.

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

`f` Array of dimension

$$\max\{1, \mathtt{nf} - \mathtt{nfsr} + \sum_{\mathtt{i=0}}^{\mathtt{nfsr-1}} \mathtt{mesh\_pts[i]}\}.$$

If no sequentially realted objectives are present, this becomes $\max\{1, \mathtt{nf}\}$.
(Output) Value of functions $f_i(x)$, $i \in I^f$, and $f_i(x,\omega)$, $\forall\omega \in \Omega^{f_i}$, $i \in I^{sr}$ at **x** at the end of execution.

`g` Array of dimension

$$\max\{1, \quad \mathtt{nineq} + \mathtt{neq} - (\mathtt{ncsrl} + \mathtt{ncsrn}) + \sum_{\mathtt{i=0}}^{\mathtt{ncsrn}} \mathtt{mesh\_pts[i + nfsr]} \\ + \sum_{\mathtt{i=0}}^{\mathtt{ncsrl}} \mathtt{mesh\_pts[i + nfsr + ncsrn]}\}.$$

If no sequentially related constraints are present, this becomes $\max\{1, \mathtt{nineq} + \mathtt{neq}\}$.
(Output) Values of all constraints at **x** at the end of execution.

`lambda` Array of dimension `nparam`$+dim$(`f`)$+dim$(`g`), where dim(`f`) and dim(`g`) denote the dimensions of the arrays `f` and `g` respectively.
(Output) Values of the Lagrange multipliers at **x** at the end of execution. They are stored in the same order as specified in the problem formulation, with those corresponding to simple bounds first (only `nparam` simple bounds could be active, thus only `nparam` multipliers are returned), next are the constraint multipliers, and finally the objective multipliers. (Note that, if appropriate, a multiplier is returned for each member of the sets of sequentially related constraints and objectives.)

`obj` **(Input)** A pointer to the user-defined function that computes the value of the objective functions $f_i(x)$ and $f_i(x,\omega)$. The detailed specification is given in § 7.1 below.

`constr` **(Input)** A pointer to the user-defined function that computes the value of the constraints. The detailed specification is given in § 7.2 below.

`gradob` **(Input)** A pointer to the function that computes the gradients of the objective functions. The user must pass the function pointer `grobfd` (declared in the header `cfsqpusr.h`) if he/she wishes that CFSQP evaluate these gradients automatically, by forward finite differences. The detailed specification is given in § 7.3 below.

gradcn **(Input)** A pointer to the function that computes the gradients of the constraints. The user must pass the function pointer `grcnfd` (declared in the header `cfsqpusr.h`) if he/she wishes that CFSQP evaluate these gradients automatically, by forward finite differences. The detailed specification is given in § 7.4 below.

5 User-Accessible Stopping Criterion and Flags

As is clear from the description of 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. A very small value of either of these two parameters may require exceedingly long execution time, depending on the complexity of the underlying problem and the nonlinearity of various functions. If the user wishes, CFSQP allows specification of three additional parameters that control three pre-selected stopping criteria via the following three globally defined variables (defined and initialized in the header file `cfsqpusr.h`): `objeps`, `objrep`, and **gLgeps**. If the user does not assign any of these values, CFSQP will never terminate on the corresponding test. CFSQP will perform corresponding checks at appropriate places during the optimization process and will terminate when either the default stopping criterion is satisfied or one of the following conditions is met:

1. **neqn** $= 0$ and $|$`fprev` $-$ `f`$| \leq$ `objeps`, where `fprev` and `f` are the value of the objective function at the previous and current iterates respectively.

2. **neqn** $= 0$ and $|$`fprev` $-$ `f`$|/|$`fprev`$| \leq$ `objrep`.

3. $\|\nabla_x L\| \leq$ **gLgeps** and $\sum_{j=1}^{n_e} |h_j(x_k)| \leq \epsilon_e$, where L is the Lagrangian.

Using one of the first two stopping criterion listed above may lead to a warning message concerning the norm of the Kuhn-Tucker vector, i.e. $\|\nabla_x L\|$, at the final iterate. This indicates that the norm is above a certain threshold, and even though the stopping criterion has been satisfied, the final iterate may not be a local minimizer.

In addition to these alternative stopping criterion, there is a globally defined logical variable **x_is_new** (declared and initialized in the header file `cfsqpusr.h`) that is initially set to `TRUE` ($= 1$) and reset to `TRUE` whenever CFSQP changes the value of x that is to be sent to one of the user-defined functions. The user may test this and do all function evaluations at once, when x is first changed, and then set **x_is_new** to `FALSE`. On subsequent calls, while **x_is_new** is still `FALSE`, the user need only return the already computed function value (remember to declare the storage set aside for this as `static` so that the data is not lost when control is returned to CFSQP.

6 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 if iprint $= 1$ or during execution if **iprint** $>$ 1:

iteration Total number of iterations (iprint $= 1$) or iteration number (**iprint** > 1).

inform See § 4. It is displayed only at the end of execution.

|Xi_k| (displayed only if ncsrl + ncsrn > 0) Total number of individual constraints from the sets of sequentially related constraints used for computation of the direction during the final iteration (i.e., $\sum_{j\in I^g_{sr}} |\Xi^{g_j}_k|$ where I^g_{sr} is the set of indices for all sets of such constraints).

|Omega_k| (displayed only if nfsr > 0) Total number of individual objectives from the sets of sequentially related objectives used for computation of the direction during the final iteration (i.e., $\sum_{i\in I^{sr}} |\Omega^{f_i}_k|$).

x Iterate.

objectives Value of objective functions $f_i(x)$, $\forall i \in I^f$, and $\Phi_{\Omega^{f_i}}(x)$, $\forall i \in I^{sr}$ at **x** (see algorithm description for definitions).

objmax (displayed only if nf > 1) The maximum value of the set of objective functions. i.e.,

$$\max\{\max_{i\in I^f} f_i(x), \max_{i\in I^{sr}} \max_{\omega\in\Omega^{f_i}} f_i(x,\omega)\}$$

or

$$\max\{\max_{i\in I^f} |f_i(x)|, \max_{i\in I^{sr}} \max_{\omega\in\Omega^{f_i}} |f_i(x,\omega)|\}$$

at **x**, depending upon the way mode was set.

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. As with objectives, only the maximum constraint value for each set of sequentially related constraints is displayed.

ncallf Number of evaluations (so far) of the individual (scalar) objective functions. Note that for $n_{f_{sr}} > 0$, every evaluation of an individual objective from a set of sequentially related objectives is considered as a function evaluation.

ncallg Number of evaluations (so far) of individual (scalar) nonlinear constraints. Once again, if $n_{sr} + \ell_{sr} > 0$, then every evaluation of an individual constraint from a set of sequentially related constraints is considered as a constraint evaluation.

d0norm Norm of the Newton direction d_k^0.

ktnorm Norm of the Kuhn-Tucker vector at the current iterate. The Kuhn-Tucker vector is given by (in its most general form, i.e., assuming sequentially related constraints and objectives are presesnt)

$$\begin{aligned}\nabla L(x_k, \zeta_k, \xi_k, \lambda_k, \mu_k, p_k) = & \sum_{j\in I^f} \zeta_{k,j} \nabla f_j(x_k) + \sum_{j\in I^{sr}} \sum_{\omega\in\Omega^{f_j}} \zeta_{k,j}^{\omega} \nabla_x f_j(x_k, \omega) + \xi_k \\ & + \sum_{j\in I_{reg}^g} \lambda_{k,j} \nabla g_j(x_k) + \sum_{j\in I_{sr}^g} \sum_{\xi\in\Xi^{g_j}} \lambda_{k,j}^{\xi} \nabla_x g_j(x_k, \xi) \\ & + \sum_{j=1}^{n_e} (\mu_{k,j} - p_{k,j}) \nabla h_j(x_k) + \sum_{j=n_e+1}^{t_e} \mu_{k,j} \nabla h_j(x_k).\end{aligned}$$

where I_{reg}^g is an index set of regular inequality constraints and I_{sr}^g is an index set for all sets of sequentially related inequality constraints.

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

For **iprint** = 3 (or $10 * N + 3$), in addition to the same information given when **iprint** = 2, the following is printed at each iteration (or at selected iterations).

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^g_k.

rhog (B $= 1$ in mode) Coefficient ρ^g_k in constructing d^g_k.

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 $i \in I^f$ and $f_i(x,\omega) - \sum_{j=1}^{n_e} p_j h_j(x)$ for $\omega \in \Omega^{f_i}$ and $i \in I^{sr}$ 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 (as defined in the user-supplied constraint function), and the corresponding values of nonlinear constraints 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 $f_i(x)$, $\forall i \in I^f$ and $f_i(x,\omega)$, $\forall \omega \in \Omega^{f_i}$, $\forall i \in I^{sr}$, 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^0_k). $\lambda_j \geq 0 \quad \forall j \in \{1,\ldots,t_i\}$ and $\mu_j \geq 0 \quad \forall j \in \{1,\ldots,t_e\}$. $\xi_i > 0$ indicates that x_i is at an upper bound and $\xi_i < 0$ indicates that x_i is at a lower bound. When (in mode) A $= 0$ and **nf** > 1 or **nfsr** > 0, $\zeta_i \geq 0$. When (in mode) A $= 1$, $\zeta_i > 0$ (resp. $\zeta_i(\omega)$) refers to $+f_i(x)$ (resp. $+f_i(x,\omega)$) and $\zeta_i < 0$ (resp. $\zeta_i(\omega) < 0$) to $-f_i(x)$ (resp. $-f_i(x,\omega)$).

hess Estimate of the Hessian matrix of the Lagrangian.

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

7 User-Supplied Functions

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

7.1 Function obj()

The function obj(), to be provided by the user, computes the value of the objective functions. If nf = 0, at least a NULL pointer to a void function must be passed (This may happen when the user is only interested in finding a feasible point). The specification of obj() for CFSQP is

```
void
obj(nparam,j,x,fj)
int nparam,j;
double *x,*fj;
{
    /*
       for given j, assign to *fj the value of the jth objective
       evaluated at x
    */
   return;
}
```

Arguments:

nparam	**(Input)** Dimension of x.
j	**(Input)** Number of the objective to be computed.
x	**(Input)** Current iterate.
fj	**(Output)** Pointer to value of the jth objective function at x.

Sequentially related objectives must always follow the regular objective function definitions. If sets of sequentially related objectives are present, a single value of j is assigned to every individual objective function in each of the sets. For instance, given a problem with two isolated objective functions and one set of sequentially related objective functions with 100 members, when CFSQP needs the value of a particular member of the sequentially related objective set, it will send the corresponding index j between j = 3 and j = 102. It is up to the user, in the function obj(), to translate this value of j into an appropriate value of $\omega \in \Omega^{f_j}$ in order to evaluate $f_j(x,\omega)$.

7.2 Function constr()

The function constr(), to be provided by the user, computes the value of the constraints. If there are no constraints, a NULL pointer to a void function should be provided anyway. The specification of constr() for CFSQP is as follows

```
void
constr(nparam,j,x,gj)
int nparam,j;
double *x,*gj;
{
   /*
       for given j, assign to *gj the value of the jth constraint
       evaluated at x
   */
   return;
}
```

Arguments:

nparam	**(Input)** Dimension of x.
j	**(Input)** Number of the constraint to be computed.
x	**(Input)** Current iterate.
gj	**(Output)** Pointer to value of the jth constraint at x.

If only isolated constraints are present, 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.

If there are sequentially related constraints present, each individual constraint is assigned its own value of j. The order of the constraints is as follows. First the nineqn − ncsrn (possibly zero) regular nonlinear inequality constraints. Next are the ncsrn (possibly zero) nonlinear sets of sequentially related constraints. As with objectives, the user must recognize a particular value of j as representing a particular constraint from a set and translate this accordingly to the appropriate $\xi \in \Xi^{g_j}$. Next are the nineq − nineqn − ncsrl (possibly zero) isolated linear inequality constraints, followed by the ncsrl (possibly zero) sets of sequentially related linear constraints. Finally, the neqn (possibly zero) nonlinear equality constraints followed by the neq − neqn (possibly zero) linear equality constraints.

7.3 Function gradob()

The function **gradob()** computes the gradients of the objective functions. The user may omit this routine and require that forward finite difference approximation be used by CFSQP via calling **grobfd()** instead (see argument gradob of CFSQP in § 4). The specification of **gradob()** for CFSQP is as follows

```
void
gradob(nparam,j,x,gradfj,dummy)
int nparam,j;
double *x,*gradfj;
void (* dummy)();
{
   /*
      assign to *gradfj a pointer to an array containing the
      gradient of the jth objective function evaluated at x
   */
   return;
}
```

Arguments:

- **nparam** **(Input)** Dimension of x.
- **j** **(Input)** Number of objective for which gradient is to be computed.
- **x** **(Input)** Current iterate.
- **gradfj** **(Output)** Pointer to array containing the gradient of the jth objective function at x.
- **dummy** **(Input)** Used by grobfd().

Note that **dummy** is passed as an arguments to **gradob** to maintain compatibility between the calling sequences of the user-defined objective gradient function and the internal CFSQP function **grobfd()** (used when forward finite difference computation of the gradient is requested by the user). The parameter j is expected to index the gradient of the objective for which the same j would index in obj().

7.4 Function gradcn()

The function **gradcn()** computes the gradients of the constraints. The user may omit this routine and require that forward finite difference approximation be used by CFSQP via calling `grcnfd()` instead (see argument `gradcn` of CFSQP in § 4). The specification of `gradcn()` for CFSQP is as follows

```
void
gradcn(nparam,j,x,gradgj,dummy)
int nparam,j;
double *x,*gradgj;
void (* dummy)();
   /*
      assign to *gradgj a pointer to an array containing the gradient
      of the jth constraint evaluated at x
   */
   return;
}
```

Arguments:

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

Once again, note that `dummy` is passed as an argument to `gradcn()` to maintain compatibility between the calling sequences of the user-defined constraint gradient function and the internal CFSQP function `grcnfd()` (used when forward finite difference computation of the gradient is requested by the user). The parameter `j` is expected to index the gradient of the constraint for which the same `j` would index in `constr()`.

8 Organization of CFSQP and Main Functions

8.1 Main Functions

CFSQP first checks for inconsistencies of input parameters using the function `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 the function `initpt()`. If necesssary, it then calls `cfsqp1()` in order to generate a point satisfying all linear constraints and nonlinear inequality constraints. Finally, CFSQP attempts to find a point satisfying the optimality conditions, again using `cfsqp1()`.

`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 CFSQP).

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

`cfsqp1` Main subroutine used possibly twice by CFSQP, first for generating a feasible iterate (as explained at the end of § 2) and second for generating an optimal iterate from that feasible iterate.

`cfsqp1` calls the following functions:

`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.

`hessian` Perform the Hessian matrix updating.

`update_omega` Called only when sequentially related constraints or objectives are present (i.e. $n_{fsr} + n_{sr} + \ell_{sr} > 0$). Updates the "active" constraint and objective sets $\Omega_k^{f_i}$ and $\Xi_k^{g_j}$.

`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 CFSQP).

`grcnfd` (optional) Compute the gradient of a constraint by forward finite differences with perturbation 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 CFSQP).

8.2 Other Functions

In addition to the QP solver QLD (see the end of §1), the following functions are used (all functions other than `cfsqp`, `grobfd`, and `grcnfd` are statically defined within CFSQP, and should not cause a clash with user-defined functions of the same name):

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

Finally, the following memory management utilities are used within CFSQP:

```
make_dv   free_dv   convert
make_iv   free_iv
make_dm   free_dm
```

9 Examples

The first problem is borrowed from [11] (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 $x \in \mathbf{R}^3$ 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.

```
#include "cfsqpusr.h"

void obj32();
void cntr32();
void grob32();
void grcn32();

int
```

```
main() {
   int i,nparam,nf,nineq,neq,mode,iprint,miter,neqn,nineqn,
       ncsrl,ncsrn,nfsr,mesh_pts[1],inform;
   double bigbnd,eps,epsneq,udelta;
   double *x,*bl,*bu,*f,*g,*lambda;

   mode=100;
   iprint=1;
   miter=500;
   bigbnd=1.e10;
   eps=1.e-8;
   epsneq=0.e0;
   udelta=0.e0;
   nparam=3;
   nf=1;
   neqn=0;
   nineqn=1;
   nineq=1;
   neq=1;
   ncsrl=ncsrn=nfsr=mesh_pts[0]=0;
   bl=(double *)calloc(nparam,sizeof(double));
   bu=(double *)calloc(nparam,sizeof(double));
   x=(double *)calloc(nparam,sizeof(double));
   f=(double *)calloc(nf+1,sizeof(double));
   g=(double *)calloc(nineq+neq+1,sizeof(double));
   lambda=(double *)calloc(nineq+neq+nf+nparam,sizeof(double));

   bl[0]=bl[1]=bl[2]=0.e0;
   bu[0]=bu[1]=bu[2]=bigbnd;

   x[0]=0.1e0;
   x[1]=0.7e0;
   x[2]=0.2e0;

   cfsqp(nparam,nf,nfsr,nineqn,nineq,neqn,neq,ncsrl,ncsrn,mesh_pts,
         mode,iprint,miter,&inform,bigbnd,eps,epsneq,udelta,bl,bu,x,
         f,g,lambda,obj32,cntr32,grob32,grcn32);
```

```
   free(bl);
   free(bu);
   free(x);
   free(f);
   free(g);
   free(lambda);
   return 0;
}
```

Following are the functions defining the objective, constraints, and their gradients.

```
void
obj32(nparam,j,x,fj)
int nparam,j;
double *x,*fj;
{
   *fj=pow((x[0]+3.e0*x[1]+x[2]),2.e0)+4.e0*pow((x[0]-x[1]),2.e0);
   return;
}

void
grob32(nparam,j,x,gradfj,dummy)
int nparam,j;
double *x,*gradfj;
void (* dummy)();
{
   double fa,fb;

   fa=2.e0*(x[0]+3.e0*x[1]+x[2]);
   fb=8.e0*(x[0]-x[1]);
   gradfj[0]=fa+fb;
   gradfj[1]=fa*3.e0-fb;
   gradfj[2]=fa;
   return;
}

void
cntr32(nparam,j,x,gj)
int nparam,j;
```

```
double *x,*gj;
{
   switch (j) {
      case 1:
         *gj=pow(x[0],3.e0)-6.e0*x[1]-4.e0*x[2]+3.e0;
         break;
      case 2:
         *gj=1.e0-x[0]-x[1]-x[2];
         break;
   }
   return;
}

void
grcn32(nparam,j,x,gradgj,dummy)
int nparam,j;
double *x,*gradgj;
void (* dummy)();
{
   switch (j) {
      case 1:
         gradgj[0]=3.e0*x[0]*x[0];
         gradgj[1]=-6.e0;
         gradgj[2]=-4.e0;
         break;
      case 2:
         gradgj[0]=gradgj[1]=gradgj[2]=-1.e0;
         break;
   }
   return;
}
```

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

```
CFSQP Version 2.0 (Released February 1994)
       Copyright (c) 1993 --- 1994
```

```
              C.T. Lawrence, J.L. Zhou
                  and A.L. Tits
                All Rights Reserved

The given initial point is feasible for inequality
         constraints and linear equality constraints:
                                 1.00000000000000e-01
                                 7.00000000000000e-01
                                 2.00000000000000e-01
 objectives
                                 7.20000000000000e+00
 constraints
                                -1.99900000000000e+00
                                 5.55111512312578e-17

 iteration                                          3
 inform                                             0
 x                              -9.86076131526265e-32
                                 0.00000000000000e+00
                                 1.00000000000000e+00
 objectives
                                 1.00000000000000e+00
 constraints
                                -1.00000000000000e+00
                                 0.00000000000000e+00
 SCV                             0.00000000000000e+00
 d0norm                          1.39452223873684e-31
 ktnorm                          1.06098265851897e-30
 ncallf                                             3
 ncallg                                             5


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

Our second example is taken from example 6 in [12]. We will use two methods to solve this problem. First we solve the problem considering all objective functions as isolated and unrelated. Next we will solve the problem considering the objectives as a single set of sequentially related functions. The problem is as follows.

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

where

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

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 $\max\limits_{i=1,\ldots,163} |f_i(x_0)| = 0.22051991555531$. A suitable main program that treats all objectives as isolated and unrelated is as follows.

```
#include "cfsqpusr.h"

void objmad();
void cnmad();

int
main() {
   int nparam,nf,nineq,neq,mode,iprint,miter,neqn,nineqn,
       ncsrl,ncsrn,nfsr,mesh_pts[1],inform;
   double bigbnd,eps,epsneq,udelta;
   double *x,*bl,*bu,*f,*g,*lambda;

   mode=111;
   iprint=1;
   miter=500;
   bigbnd=1.e10;
   eps=1.e-8;
   epsneq=0.e0;
   udelta=0.e0;
   nparam=6;
```

```
    nf=163;
    neqn=0;
    nineqn=0;
    nineq=7;
    neq=0;
    ncsrl=ncsrn=nfsr=mesh_pts[0]=0;
    bl=(double *)calloc(nparam,sizeof(double));
    bu=(double *)calloc(nparam,sizeof(double));
    x=(double *)calloc(nparam,sizeof(double));
    f=(double *)calloc(nf+1,sizeof(double));
    g=(double *)calloc(nineq+neq+1,sizeof(double));
    lambda=(double *)calloc(nineq+neq+nf+nparam+1,sizeof(double));

    bl[0]=bl[1]=bl[2]=bl[3]=bl[4]=bl[5]=-bigbnd;
    bu[0]=bu[1]=bu[2]=bu[3]=bu[4]=bu[5]=bigbnd;

    x[0]=0.5e0;
    x[1]=1.e0;
    x[2]=1.5e0;
    x[3]=2.e0;
    x[4]=2.5e0;
    x[5]=3.e0;

    cfsqp(nparam,nf,nfsr,nineqn,nineq,neqn,neq,ncsrl,ncsrn,mesh_pts,
          mode,iprint,miter,&inform,bigbnd,eps,epsneq,udelta,bl,bu,x,
          f,g,lambda,objmad,cnmad,grobfd,grcnfd);

    free(bl);
    free(bu);
    free(x);
    free(f);
    free(g);
    free(lambda);
    return 0;
}
```

We choose to compute the gradients of functions by means of finite difference approximation. Thus only functions that define the objectives and constraints are needed as follows. (These functions will not change when we consider the problem as a sequentially related set of objectives, hence we only list them once.)

```
void
objmad(nparam,j,x,fj)
int nparam,j;
double *x,*fj;
{
   double pi,theta;
   int i;

   pi=3.14159265358979e0;
   theta=pi*(8.5e0+j*0.5e0)/180.e0;
   *fj=0.e0;
   for (i=0; i<=5; i++)
      *fj=*fj+cos(2.e0*pi*x[i]*sin(theta));
   *fj=2.e0*(*fj+cos(2.e0*pi*3.5e0*sin(theta)))/15.e0
      +1.e0/15.e0;
   return;
}

void
cnmad(nparam,j,x,gj)
int nparam,j;
double *x,*gj;
{
   double ss;

   ss=0.425e0;
   switch (j) {
      case 1:
         *gj=ss-x[0];
         break;
      case 2:
         *gj=ss+x[0]-x[1];
         break;
      case 3:
         *gj=ss+x[1]-x[2];
         break;
      case 4:
```

```
        *gj=ss+x[2]-x[3];
        break;
     case 5:
        *gj=ss+x[3]-x[4];
        break;
     case 6:
        *gj=ss+x[4]-x[5];
        break;
     case 7:
        *gj=ss+x[5]-3.5e0;
        break;
  }
  return;
}
```

After running the first 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)

```
     CFSQP Version 2.0 (Released February 1994)
            Copyright (c) 1993 --- 1994
             C.T. Lawrence, J.L. Zhou
                  and A.L. Tits
               All Rights Reserved

  The given initial point is feasible for inequality
           constraints and linear equality constraints:
                                5.00000000000000e-01
                                1.00000000000000e+00
                                1.50000000000000e+00
                                2.00000000000000e+00
                                2.50000000000000e+00
                                3.00000000000000e+00
   objmax                       2.20519865065595e-01
   constraints
                               -7.50000000000000e-02
                               -7.50000000000000e-02
                               -7.50000000000000e-02
                               -7.50000000000000e-02
```

```
                          -7.50000000000002e-02
                          -7.50000000000002e-02
                          -7.50000000000002e-02

iteration                                     7
inform                                        0
x                          4.25000000000000e-01
                           8.50000000000000e-01
                           1.27500000000000e+00
                           1.70000000000000e+00
                           2.18407631966882e+00
                           2.87327550964478e+00
objective max4             1.14218413252211e-01
objmax                     1.13104727498258e-01
constraints
                           0.00000000000000e+00
                           0.00000000000000e+00
                           0.00000000000000e+00
                           0.00000000000000e+00
                          -5.90763196688169e-02
                          -2.64199189975961e-01
                          -2.01724490355223e-01
SCV                        0.00000000000000e+00
d0norm                     1.56621622756395e-10
ktnorm                     2.05641104350305e-11
ncallf                                     1141


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

We now list the appropriate modifications of the above main program that will tell CFSQP to exploit the structure of the problem and treat the objectives as a sequentially related set. Thus, CFSQP will use the algorithm FSQP-SR. The majority of the code remains unchanged, we list here the initialization portion only. Note that the parameters **nf**, **nfsr**, and **mesh_pts[0]** have been changed. Note also that the amount of memory allocated for the appropriate arrays has changed.

```
mode=111;
iprint=1;
miter=500;
bigbnd=1.e10;
eps=1.e-8;
epsneq=0.e0;
udelta=0.e0;
nparam=6;
nf=1;                   /* One SR objective set with 163     */
nfsr=1;                 /* sequentially related members.     */
mesh_pts[0]=163;
neqn=0;
nineqn=0;
nineq=7;
neq=0;
ncsrl=ncsrn=0;
bl=(double *)calloc(nparam,sizeof(double));
bu=(double *)calloc(nparam,sizeof(double));
x=(double *)calloc(nparam,sizeof(double));
f=(double *)calloc(mesh_pts[0]+1,sizeof(double));
g=(double *)calloc(nineq+neq+1,sizeof(double));
lambda=(double *)calloc(nineq+neq+mesh_pts[0]+nparam+1,
                        sizeof(double));
```

After running the problem using algorithm FSQP-SR on a SUN 4/SPARC station 1, the following output is obtained

```
CFSQP Version 2.0 (Released February 1994)
       Copyright (c) 1993 --- 1994
        C.T. Lawrence, J.L. Zhou
             and A.L. Tits
          All Rights Reserved

The given initial point is feasible for inequality
        constraints and linear equality constraints:
                          5.00000000000000e-01
                          1.00000000000000e+00
```

	1.50000000000000e+00
	2.00000000000000e+00
	2.50000000000000e+00
	3.00000000000000e+00
objectives	
	2.20519865065595e-01
objmax	2.20519865065595e-01
constraints	
	-7.50000000000000e-02
	-7.50000000000000e-02
	-7.50000000000000e-02
	-7.50000000000000e-02
	-7.50000000000002e-02
	-7.50000000000002e-02
	-7.50000000000002e-02
iteration	7
inform	0
|Omega_k|	7
x	4.25000000000000e-01
	8.50000000000000e-01
	1.27500000000000e+00
	1.70000000000000e+00
	2.18407631958035e+00
	2.87327550951555e+00
objectives	
	1.13104727457934e-01
objective max4	1.13659863034301e-01
objmax	1.13104727457934e-01
constraints	
	0.00000000000000e+00
	0.00000000000000e+00
	0.00000000000000e+00
	0.00000000000000e+00
	-5.90763195803512e-02
	-2.64199189935201e-01
	-2.01724490484449e-01
SCV	0.00000000000000e+00

```
d0norm                    4.33218838740540e-12
ktnorm                    2.09600754392242e-12
ncallf                                    1141


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

We should mention that it is actually unusual to get the same results for both methods in terms of function evaluations and number of iterations. Usually, computing the search direction based on a small subset of objectives and constraints causes an increase in the number of iterations and function evaluations. For this example, since only a small subset of the objectives were used to construct the QP subproblems at each iteration, algorithm FSQP-SR executed in significantly less time, and required far fewer gradient evaluations.

Our third example is borrowed from [11] (Problem 71). It involves both equality and inequality nonlinear constraints and is defined by

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

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 CFSQP to solve this problem is given below.

```
#include "cfsqpusr.h"

void obj71();
void cntr71();
void grob71();
void grcn71();

int
main() {
   int nparam,nf,nineq,neq,mode,iprint,miter,neqn,nineqn,
       ncsrl,ncsrn,nfsr,mesh_pts[1],inform;
   double bigbnd,eps,epsneq,udelta;
   double *x,*bl,*bu,*f,*g,*lambda;
```

```
mode=100;
iprint=1;
miter=500;
bigbnd=1.e10;
eps=1.e-7;
epsneq=7.e-6;
udelta=0.e0;
nparam=4;
nf=1;
neqn=1;
nineqn=1;
nineq=1;
neq=1;
ncsrl=ncsrn=nfsr=mesh_pts[0]=0;
bl=(double *)calloc(nparam,sizeof(double));
bu=(double *)calloc(nparam,sizeof(double));
x=(double *)calloc(nparam,sizeof(double));
f=(double *)calloc(nf+1,sizeof(double));
g=(double *)calloc(nineq+neq+1,sizeof(double));
lambda=(double *)calloc(nineq+neq+nf+nparam+1,sizeof(double));

bl[0]=bl[1]=bl[2]=bl[3]=1.e0;
bu[0]=bu[1]=bu[2]=bu[3]=5.e0;


x[0]=1.e0;
x[1]=5.e0;
x[2]=5.e0;
x[3]=1.e0;

cfsqp(nparam,nf,nfsr,nineqn,nineq,neqn,neq,ncsrl,ncsrn,mesh_pts,
      mode,iprint,miter,&inform,bigbnd,eps,epsneq,udelta,bl,bu,x,
      f,g,lambda,obj71,cntr71,grob71,grcn71);

free(bl);
free(bu);
free(x);
free(f);
```

```
    free(g);
    free(lambda);
    return 0;
}
```

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

```
void
obj71(nparam,j,x,fj)
int nparam,j;
double *x,*fj;
{
    *fj=x[0]*x[3]*(x[0]+x[1]+x[2])+x[2];
    return;
}

void
grob71(nparam,j,x,gradfj,dummy)
int nparam,j;
double *x,*gradfj;
void (* dummy)();
{
    gradfj[0]=x[3]*(x[0]+x[1]+x[2])+x[0]*x[3];
    gradfj[1]=x[0]*x[3];
    gradfj[2]=x[0]*x[3]+1.e0;
    gradfj[3]=x[0]*(x[0]+x[1]+x[2]);
    return;
}

void
cntr71(nparam,j,x,gj)
int nparam,j;
double *x,*gj;
{
    switch (j) {
        case 1:
            *gj=25.e0-x[0]*x[1]*x[2]*x[3];
            break;
        case 2:
```

```
        *gj=x[0]*x[0]+x[1]*x[1]+x[2]*x[2]+x[3]*x[3]-40.e0;
        break;
   }
   return;
}

void
grcn71(nparam,j,x,gradgj,dummy)
int nparam,j;
double *x,*gradgj;
void (* dummy)();
{
   switch (j) {
      case 1:
         gradgj[0]=-x[1]*x[2]*x[3];
         gradgj[1]=-x[0]*x[2]*x[3];
         gradgj[2]=-x[0]*x[1]*x[3];
         gradgj[3]=-x[0]*x[1]*x[2];
         break;
      case 2:
         gradgj[0]=2.e0*x[0];
         gradgj[1]=2.e0*x[1];
         gradgj[2]=2.e0*x[2];
         gradgj[3]=2.e0*x[3];
         break;
   }
   return;
}
```

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

```
   CFSQP Version 2.0 (Released February 1994)
          Copyright (c) 1993 --- 1994
           C.T. Lawrence, J.L. Zhou
                and A.L. Tits
              All Rights Reserved

The given initial point is feasible for inequality
```

```
        constraints and linear equality constraints:
                                 1.00000000000000e+00
                                 5.00000000000000e+00
                                 5.00000000000000e+00
                                 1.00000000000000e+00
objectives
                                 1.60000000000000e+01
constraints
                                 0.00000000000000e+00
                                -1.20000000000000e+01

iteration                                           8
inform                                              0
x                                1.00000000000000e+00
                                 4.74299965181119e+00
                                 3.82114996517964e+00
                                 1.37940829580298e+00
objectives
                                 1.70140172891577e+01
constraints
                                -3.51718654201250e-12
                                -3.51008111465489e-12
SCV                              3.51008111465489e-12
d0norm                           2.39563998677879e-08
ktnorm                           3.40098916281420e-08
ncallf                                              9
ncallg                                             31


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

Our fourth example is borrowed from [13] (Problem TP374). It involves three sets of sequentially related nonlinear inequality constraints and, given an integer r, is defined by

$$
\begin{array}{cll}
\min\limits_{x\in\mathbf{R}^{10}} & x_{10} & \\
\text{s.t.} & z(t_i)-(1-x_{10})^2 \geq 0, & i=1,\ldots,r \\
& (1+x_{10})^2 - z(t_i) \geq 0, & i=r+1,\ldots,2r \\
& x_{10}^2 - z(t_i) \geq 0, & i=2r+1,\ldots,3.5r,
\end{array}
$$

where

$$z(t) = \left(\sum_{k=1}^{9} x_k \cos(kt)\right)^2 + \left(\sum_{k=1}^{9} x_k \sin(kt)\right)^2$$

and

$$t_i = \begin{cases} \pi(i-1)0.025 & i = 1,\ldots,r \\ \pi(i-1-r)0.025 & i = r+1,\ldots,2r \\ \pi(1.2+(i-1-2r)0.2)0.25 & i = 2r+1,\ldots,3.5r. \end{cases}$$

We let $r = 100$ and use the feasible initial guess: $x_0 = (0.1,\ldots,0.1,1)^T$ with the corresponding value of the objective function $f(x_0) = 1$. We will use the algorithm FSQP-SR to solve this problem. A suitable program that invokes CFSQP to do this is given below. Notice that even though we really have 350 constraints, they are interpreted as being in 3 sets of sequentially related constraints. Hence, `nineq = nineqn = 3` and not 350.

```
#include "cfsqpusr.h"

#define r 100

void obj();
void cntr();
void grob();

int
main() {
   int i,nparam,nf,nineq,neq,mode,iprint,miter,neqn,nineqn,
       ncsrl,ncsrn,nfsr,mesh_pts[3],numc,inform;
   double bigbnd,eps,epsneq,udelta;
   double *x,*bl,*bu,*f,*g,*lambda;

   mode=100;
   iprint=1;
   miter=500;
   bigbnd=1.e10;
   eps=1.e-7;
   epsneq=0.e0;
   udelta=0.e0;
   nparam=10;
   nf=1;
   neqn=0;
```

```
    nineqn=nineq=ncsrn=3;
    ncsrl=0;
    mesh_pts[0]=mesh_pts[1]=r;
    mesh_pts[2]=3*r/2;
    neq=nfsr=0;
    numc=3.5*r;
    bl=(double *)calloc(nparam,sizeof(double));
    bu=(double *)calloc(nparam,sizeof(double));
    x=(double *)calloc(nparam,sizeof(double));
    f=(double *)calloc(nf+1,sizeof(double));
    g=(double *)calloc(numc+1,sizeof(double));
    lambda=(double *)calloc(numc+nf+nparam+1,sizeof(double));

    bl[0]=bl[1]=bl[2]=bl[3]=bl[4]=bl[5]=bl[6]=bl[7]=bl[8]=bl[9]=-bigbnd;
    bu[0]=bu[1]=bu[2]=bu[3]=bu[4]=bu[5]=bu[6]=bu[7]=bu[8]=bu[9]=bigbnd;

    x[0]=x[1]=x[2]=x[3]=x[4]=x[5]=x[6]=x[7]=x[8]=0.1e0;
    x[9]=1.e0;

    cfsqp(nparam,nf,nfsr,nineqn,nineq,neqn,neq,ncsrl,ncsrn,mesh_pts,
          mode,iprint,miter,&inform,bigbnd,eps,epsneq,udelta,bl,bu,x,
          f,g,lambda,obj,cntr,grob,grcnfd);

    free(bl);
    free(bu);
    free(x);
    free(f);
    free(g);
    free(lambda);
    return 0;
}
```

Following are the functions that define the objective, constraints and the objective gradients. We use finite difference approximations for the gradients of the constraints. Note that in the constraint evaluation function the constraints, even though they are all of the same type, must be ordered according to the ordering within the constraint "sets." The sets themselves must be ordered as determined by the `mesh_pts[]` array.

```
void
```

```
obj(nparam,j,x,fj)
int nparam,j;
double *x,*fj;
{
   *fj=x[9];
   return;
}

void
grob(nparam,j,x,gradfj,dummy)
int nparam,j;
double *x,*gradfj;
void (* dummy)();
{
   gradfj[0]=0.e0;
   gradfj[1]=0.e0;
   gradfj[2]=0.e0;
   gradfj[3]=0.e0;
   gradfj[4]=0.e0;
   gradfj[5]=0.e0;
   gradfj[6]=0.e0;
   gradfj[7]=0.e0;
   gradfj[8]=0.e0;
   gradfj[9]=1.e0;
   return;
}

void
cntr(nparam,j,x,gj)
int nparam,j;
double *x,*gj;
{
   double t,z,s1,s2;
   int k;

   s1=s2=0.e0;
   if (j<=r) t=3.14159265e0*(j-1)*0.025e0;
   else {
```

```
      if (j<=2*r) t=3.14159265e0*(j-1-r)*0.025e0;
      else t=3.14159265e0*(1.2e0+(j-1-2*r)*0.2e0)*0.25e0;
   }
   for (k=1; k<=9; k++) {
      s1=s1+x[k-1]*cos(k*t);
      s2=s2+x[k-1]*sin(k*t);
   }
   z=s1*s1 + s2*s2;
   if (j<=r) *gj=(1.e0-x[9])*(1.e0-x[9])-z;
   else {
      if (j<=2*r) *gj=z-(1.e0+x[9])*(1.e0+x[9]);
      else *gj=z-x[9]*x[9];
   }
   return;
}
```

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

```
   CFSQP Version 2.0 (Released February 1994)
          Copyright (c) 1993 --- 1994
           C.T. Lawrence, J.L. Zhou
                and A.L. Tits
             All Rights Reserved

The given initial point is feasible for inequality
         constraints and linear equality constraints:
                          1.00000000000000e-01
                          1.00000000000000e-01
                          1.00000000000000e-01
                          1.00000000000000e-01
                          1.00000000000000e-01
                          1.00000000000000e-01
                          1.00000000000000e-01
                          1.00000000000000e-01
                          1.00000000000000e-01
                          1.00000000000000e+00
 objectives
                          1.00000000000000e+00
```

```
constraints
                              -1.28661830051682e-04
                              -3.19000000000000e+00
                              -1.90000000000000e-01

iteration                                        47
inform                                            0
|Xi_k|                                           30
x                              4.99999999997398e-01
                              -8.29629818539665e-12
                               1.20042143033773e-11
                               1.23097556828655e-11
                               4.55573581501274e-12
                              -1.11252133627631e-11
                              -1.23383689157888e-12
                               1.99819837204242e-12
                               5.20795089164941e-14
                               5.00000002485458e-01
objectives
                               5.00000002485458e-01
constraints
                              -2.44785253178392e-09
                              -2.00000000743848e+00
                              -2.46756859390018e-09
SCV                            0.00000000000000e+00
d0norm                         2.85515842346338e-09
ktnorm                         1.21313837573104e-09
ncallf                                           47
ncallg                                        24078


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

Our fifth and final example is borrowed from [14] (Problem 2) and is an example of a discretized semi-infinite program. The original semi-infinite programming problem is defined by

$$\begin{array}{ll}\min\limits_{x\in\mathbf{R}^2} & \frac{1}{3}x_1^2 + x_2^2 + \frac{1}{2}x_1 \\ \text{s.t.} & x_2^2 - x_2 + x_1t^2 - (1 - x_1^2t^2)^2 \geq 0 \quad \forall\, t \in [0,1].\end{array}$$

In order to use the algorithm, we must choose a finite subset $\Xi \subset [0,1]$ (using the notation introduced in § 3). There are, of course, many ways that we could do this. A suitable choice for Ξ, given a level of discretization q, is the *uniform* discretization

$$\Xi = \left\{0, \frac{1}{q}, \frac{2}{q}, \ldots, \frac{(q-1)}{q}, 1\right\}.$$

Thus we have the new discretized SIP problem with finitely many constraints

$$\begin{array}{ll}\min\limits_{x\in\mathbf{R}^2} & \frac{1}{3}x_1^2 + x_2^2 + \frac{1}{2}x_1 \\ \text{s.t.} & x_2^2 - x_2 + x_1\xi^2 - (1 - x_1^2\xi^2)^2 \geq 0 \quad \forall\, \xi \in \Xi.\end{array}$$

We chose $q = 100$ and the feasible initial guess: $x_0 = (-1,-2)^T$ with corresponding value of the objective function $f(x_0) = 3.833333$. The final solution of the original problem is: $x^* = (-0.75, -0.618034)^T$ with $f(x^*) = 0.194466$ and the constraint is active only at the point $t = 0$. Though the normal algorithm will solve the problem, algorithm FSQP-SR is specifically designed to exploit the structure the problem. A suitable main program that will call CFSQP to do this is as follows:

```
#include "cfsqpusr.h"

void obj();
void cntr();
void grob();
void grcn();

int
main() {
   int i,nparam,nf,nineq,neq,mode,iprint,miter,neqn,nineqn,
       ncsrl,ncsrn,nfsr,mesh_pts[1],inform;
   double bigbnd,eps,epsneq,udelta;
   double *x,*bl,*bu,*f,*g,*lambda;

   mode=100;
   iprint=1;
   miter=500;
```

```
bigbnd=1.e10;
eps=1.e-4;
epsneq=0.e0;
udelta=0.e0;
nparam=2;
nf=1;
neqn=0;
nineqn=nineq=1;
ncsrn=1;
ncsrl=0;
mesh_pts[0]=501;
neq=nfsr=0;
bl=(double *)calloc(nparam,sizeof(double));
bu=(double *)calloc(nparam,sizeof(double));
x=(double *)calloc(nparam,sizeof(double));
f=(double *)calloc(nf+1,sizeof(double));
g=(double *)calloc(nineq+(mesh_pts[0]-1)*(ncsrl+ncsrn)+neq+1,
   sizeof(double));
lambda=(double *)calloc(nineq+(mesh_pts[0]-1)*(ncsrl+ncsrn)+neq+
       nf+nparam,sizeof(double));

bl[0]=bl[1]=-bigbnd;
bu[0]=bu[1]=bigbnd;

x[0]=-1.e0;
x[1]=-2.e0;

cfsqp(nparam,nf,nfsr,nineqn,nineq,neqn,neq,ncsrl,ncsrn,mesh_pts,
      mode,iprint,miter,&inform,bigbnd,eps,epsneq,udelta,bl,bu,x,
      f,g,lambda,obj,cntr,grob,grcn);

free(bl);
free(bu);
free(x);
free(f);
free(g);
free(lambda);
return 0;
```

```
}
```

Following are the functions that define the objective, constraint, and their gradients.

```
void
obj(nparam,j,x,fj)
int nparam,j;
double *x,*fj;
{
   *fj=(1.e0/3.e0)*pow(x[0],2.e0)+pow(x[1],2.e0)+0.5e0*x[0];
   return;
}

void
grob(nparam,j,x,gradfj,dummy)
int nparam,j;
double *x,*gradfj;
void (* dummy)();
{
   gradfj[0]=(2.e0/3.e0)*x[0]+0.5e0;
   gradfj[1]=2.e0*x[1];
   return;
}

void
cntr(nparam,j,x,gj)
int nparam,j;
double *x,*gj;
{
   double y;

   y=(j-1)/100.e0;
   *gj=pow((1.e0-pow(y*x[0],2.e0)),2.e0)-x[0]*y*y-pow(x[1],2.e0)
       +x[1];
   return;
}

void
grcn(nparam,j,x,gradgj,dummy)
```

```
int nparam,j;
double *x,*gradgj;
void (* dummy)();
{
   double y;

   y=(j-1)/100.e0;
   gradgj[0]=-4.e0*(1.e0-pow(y*x[0],2.e0))*y*y*x[0]-y*y;
   gradgj[1]=-2.e0*x[1]+1.e0;
   return;
}
```

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

```
   CFSQP Version 2.0 (Released February 1994)
          Copyright (c) 1993 --- 1994
           C.T. Lawrence, J.L. Zhou
                and A.L. Tits
              All Rights Reserved

The given initial point is feasible for inequality
         constraints and linear equality constraints:
                            -1.00000000000000e+00
                            -2.00000000000000e+00
 objectives
                             3.83333333333333e+00
 constraints
                            -5.00000000000000e+00

 iteration                                      6
 inform                                         0
 |Xi_k|                                         2
 x                          -7.49999999972904e-01
                            -6.18033989004395e-01
 objectives
                             1.94466011564684e-01
 constraints
                            -5.69078895118480e-10
```

```
SCV                  0.00000000000000e+00
d0norm               2.55928300682528e-10
ktnorm               2.54322186261339e-10
ncallf                                  6
ncallg                               3073


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

10 Results for Test Problems

These results are provided for the user to compare CFSQP with his/her favorite code (see also [2–4,6]) and were all obtained with C = 1 in `mode`. The results listed in the first three tables were all obtained without the use of the algorithm FSQP-SR. Table 1 contains results obtained for some non-minimax test problems from [11] (the same initial points as in [11] were selected). **prob** indicates the problem number as in [11], **nparam** the number of free variables, **nineqn** the number of nonlinear (inequality) 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 function, `d0norm` the norm of SQP direction at the final iterate, and **eps** the norm requirement for the SQP direction (in the stopping criterion). On each test problem, **eps** was selected so as to achieve the same field precision as in [11]. 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 [15]; `cb2`, `cb3`, `r-s`, **wong** and **colv** are from [16; Examples 5.1-5] (more recent results on problems `bard` down to **wong** can be found in [17]); **kiw1** and **kiw4** are from [18] (results for **kiw2** and **kiw3** are not reported due to data disparity); **mad1** to **mad8** are from [12, Examples 1-8]; **polk1** to **polk4** are from [19]. Some of these test problems allow one to freely select the number of variables; problems **wat6** and **wat20** correspond to 6 and 20 variables respectively, and **mad810**, **mad830** and **mad850** 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 [11] 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, **nparam**, **nineqn**, **ncallf**,**ncallg**, **iter**, **d0norm** and **eps** are as in Table 1 (but ncallf is the total number of evaluations of *scalar* objective functions). **nf** is the number of objective functions in the max, and **objmax** is the final value of the max of the objective functions.

Table 3 contains results of problems with nonlinear equality constraints from [11]. Most columns are the same as described before. eps is not displayed in the table as it is set to 10^{-4} for all of the problems. epseqn is the norm requirement on the values of the equality constraints and is chosen close to the corresponding values in [11]. 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.

Table 4 contains the results for problems with a set (or sets) of sequentially related contraints solved via the algorithm FSQP-SR. All problems in this table are discretized semi-infinite programs. Problems cw_2 through cw_7 are borrowed from [14], **hu_1** through **hu_12** are from [20], hz_1 is from [21], oet_1 through oet_7 are from [22], pt_1 is from [23], and **sch_3** is from [13]. Columns prob, B, **ncallf**, **ncallg**, **iter**, **objective**, and **d0norm** are as before. **ncsrl** is the number of linear sequentially related constraint sets, while **ncsrn** is the number of nonlinear sequentially related constraint sets. $\sum|\Xi^{g_i}|$ indicates the total number of constraints for the problem, i.e. the sum of the number of members in each constraint set. Finally, $|\Xi^*|$ is the total number of individual constraints used to construct the search direction during the final iteration. As in Table 3, we do not include **eps** in the table since it was set to 10^{-4} for all problems.

Table 5 contains results for problems with sets of sequentially related objective functions solved using the algorithm FSQP-SR. Most columns are as before, except **nfsr** is the number of sequentially related objective sets, $\sum|\Omega^{f_i}|$ is the total number of objective functions, and $|\Omega^*|$ is the number of objective functions used to construct the search direction during the final iteration. Once again, the norm requirement for the SQP direction, **eps**, was set to 10^{-4} for all problems. All problems except for **sch_u3**, which is taken from [13], are the same as the corresponding problems in Table 4, except that they are rewritten in minimax form.

In other words, in the original reference they were posed in the form:

$$\begin{array}{ll} \min\limits_{x\in\mathbf{R}^{n+1}} & x_{n+1} \\ \text{s.t.} & f_i(x,\omega) - x_{n+1} \leq 0 \quad \forall\omega \in \Omega^{f_i} \quad i = 1,\ldots,n_{fsr}. \end{array}$$

For this table, we equivalently reformulate the problems as:

$$\min_{x\in\mathbf{R}^n} \max_{1\leq i\leq n_{fsr}} \max_{\omega\in\Omega^{f_i}} f_i(x,\omega).$$

Additionally, problems `oet_1m` to `oet_7m` have absolute value objective functions, i.e. `A`= 1 in **mode**. Finally, as before, we do not list the value of **eps** used for the stopping criterion, as it was set to 10^{-4} for all problems.

11 Programming Tips

In both FSQP-AL and FSQP-NL, at each trial point in the arc search, evaluation of objectives/constraints is discontinued as soon as it has been found that one of the inequalities in the arc search test fails (see § 2). Other than a minor exception (see again § 2), objectives/constraints within a given type (linear equalities, linear inequalities, nonlinear inequalities, nonlinear equalities, objectives) are evaluated in the order they are defined in the user supplied subroutines. In consequence, the CPU-wise user will place earlier in his/her list the functions whose evaluation is less expensive.

The order in which CFSQP evaluates the various objectives and constraints during the line search varies from trial point to trial point, as the functions deemed more likely to cause rejection of the trial steps are evaluated first. On the other hand, in many applications, it is far more efficient to evaluate all (or at least more than one) of the objectives and constraints concurrently, as they are all obtained as byproducts of expensive simulations (e.g., involving finite element computation). This situation can be accomodated by making use of the flag **x_is_new** as outlined in § 5. If gradients are computed by finite difference, however, this will not be of much help, as `x_is_new` will be set to `TRUE` by CFSQP with every perturbation of a component of x. In such a case, the user may want to save (in **static** arrays) the last **nparam**+1 values of x and the corresponding objective/constraint function values. Then, whenever a function evaluation is requested by CFSQP, first check whether the same value of **x** has already been used. If so, entirely bypass the expensive simulation.

12 Portability

The CFSQP source code was designed to be as portable as possible. In order to satisfy a broad range of users, the distributed source code contains both Kernighan & Richie and

ANSI compliant function definitions and prototypes. The users need not concern themselves with which of these standards will be used, as the correct definitions and prototypes will automatically be selected by the user's pre-compiler (all ANSI specific definitions, etc. are separated via an `#ifdef __STDC__`).

13 Trouble-Shooting

It is important to keep in mind some limitations of CFSQP. First, similar to most codes targeted at smooth problems, it is likely to encounter difficulties when confronted to nonsmooth functions such as, for example, functions involving matrix eigenvalues. Second, because CFSQP generates feasible iterates, it may be slow if the feasible set is very "thin" or oddly shaped. Third, concerning equality constraints, 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 strictly feasible set defined by $h_j(x) < 0$ for such j is empty. This may cause difficulties for CFSQP because, in CFSQP, $h_j(x) = 0$ 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.

A common failure mode for CFSQP, corresponding to `inform` = 5 or 6, is that of the QP solver in constructing `d0` or `d1`. This is often due to linear dependence (or almost dependence) of gradients of equality constraints or active inequality constraints. Sometimes this problem can be circumvented by making use of a more robust (but likely slower) QP solver. The user may also want to check the Jacobian matrix and identify which constraints are the culprit. Eliminating redundant constraints or formulating the constraints differently (without changing the feasible set) may then be the way to go.

Finally, when CFSQP fails in the line search (`inform`=4), it is typically due to inaccurate computation of the search direction. Two possible reasons are: (i) Insufficient accuracy of the QP solver; again, it may be appropriate to substitute a different QP solver. (ii) Insufficient accuracy of gradient computation, e.g., when gradients are computed by finite differences. A remedy may be to provide analytical gradients or, more astutely, to resort to "automatic differentiation".

14 Acknowledgments

The authors are indebted to Dr. E.R. Panier for many invaluable comments and suggestions. Additionally, we would like to thank the many people who used previous versions of CFSQP and took the time to point out problems and potential difficulties to us. There are a few who deserve special recognition for going well above and beyond the call of duty. These include Greg Anderson of Intel, Martin Wauchope of CRA, Australia, and Steven Drucker of MIT.

Their efforts went a long way to improving the robustness and portability of the CFSQP code. Thanks to all!

15 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* 59 (1993) , 261–276.

[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 (1993) , 455–476.

[5] J.L. Zhou & A.L. Tits, "An SQP Algorithm for Finely Discretized Continuous Minimax Problems and Other Minimax Problems With Many Objective Functions," Institute for Systems Research, University of Maryland, TR 93-83, College Park, Maryland 20742, 1993.

[6] J.L. Zhou & A.L. Tits, "An SQP Algorithm for Finely Discretized SIP Problems and Other Problems with Many Constraints," Institute for Systems Research, University of Maryland, in preparation, College Park, Maryland 20742, 1993.

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

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

[9] S. I. Feldman, D. M. Gay, M. W. Maimone & N. L. Schryer, "A Fortran to C Converter," AT & T Bell Laboratories, Computer Science Technical Report No. 149, 1990.

[10] 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.

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

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

[13] K. Schittkowski, "Solving Nonlinear Programming Problems with Very Many Constraints," Mathematisches Institut, Universität Bayreuth, Report No. 294, Bayreuth, Germany, 1991.

[14] I.D. Coope & G.A. Watson, "A Projected Lagrangian Algorithm for Semi-Infinite Programming," *Math. Programming* 32 (1985) , 337–356.

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

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

[17] 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 .

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

[19] 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*, Tampa, Florida (December 1989) .

[20] M. Huth, "Superlinear konvergente Verfahren zur Lösung semi-infiniter Optimierungsaufgaben," Pädagog. Hochsch. Halle, Ph.D. dissertation, 1987 .

[21] R. Hettich & P. Zencke, *Numerische Methoden der Approximation und Semi-Infiniten Optimierung*, Teubner Studienbücher, 1982.

[22] K. Oettershagen, *Ein Superlinear Konvergenter Algorithmus zur Losung Semi-Infiniter Optimierungsprobleme*, Ph.D. Thesis, Bonn University, 1982.

[23] E.R. Panier & A.L. Tits, "A Globally Convergent Algorithm with Adaptively Refined Discretization for Semi-Infinite Optimization Problems Arising in Engineering Design," *IEEE Trans. Automat. Control* AC-34 (1989) , 903–908.

prob	B	nparam	nineqn	ncallf	ncallg	iter	objective	d0norm	eps
p12	0	2	1	7	14	7	−.300000000E+02	.17E-06	.10E-05
	1			7	12	7	−.300000000E+02	.19E-06	.10E-05
p29	0	3	1	11	20	10	−.226274170E+02	.51E-06	.10E-04
	1			11	15	10	−.226274170E+02	.25E-05	.10E-04
p30	0	3	1	18	35	18	.100000000E+01	.41E-07	.10E-06
	1			24	24	24	.100000000E+01	.62E-07	.10E-06
p31	0	3	1	9	10	7	.600000000E+01	.10E-05	.10E-04
	1			9	17	9	.600000000E+01	.22E-05	.10E-04
p32	0	3	1	3	5	3	.100000000E+01	.14E-30	.10E-07
	1			3	4	3	.100000000E+01	.0	.10E-07
p34	0	3	2	7	28	7	−.834032443E+00	.19E-08	.10E-07
	1			9	24	9	−.834032445E+00	.39E-11	.10E-07
p43	0	4	3	10	46	8	−.440000000E+02	.71E-05	.10E-04
	1			11	46	11	−.440000000E+02	.11E-05	.10E-04
p44	0	4	0	6	0	6	−.150000000E+02	.0	.10E-07
	1			6		6	−.150000000E+02	.0	.10E-07
p51	0	5	0	8	0	6	.193107145E−15	.46E-06	.10E-05
	1			9		8	.229528403E−17	.15E-08	.10E-05
p57	0	2	1	7	5	3	.306463061E−01	.26E-05	.10E-04
	1			7	5	3	.306463061E−01	.26E-05	.10E-04
p66	0	3	2	8	30	8	.518163274E+00	.19E-08	.10E-07
	1			9	24	9	.518163274E+00	.53E-08	.10E-07
p67	0	3	14	21	305	21	−.116211927E+02	.23E-05	.10E-04
	1			62	868	62	−.116211927E+02	.25E-05	.10E-04
p70	0	4	1	62	36	32	.940197325E−02	.12E-07	.10E-06
	1			42	42	38	.940197325E−02	.84E-07	.10E-06
p76	0	4	0	6	0	6	−.468181818E+01	.16E-04	.10E-03
	1			6		6	−.468181818E+01	.16E-04	.10E-03
p84	0	5	6	4	30	4	−.528033513E+07	.0	.10E-07
	1			4	29	4	−.528033513E+07	.19E-14	.10E-07
p85	0	5	38	45	1781	45	−.240504370E+01	.32E-03	.10E-02
	1			28	1064	28	−.173019329E+01	.23E-03	.10E-02
p86	0	5	0	7	0	5	−.323486790E+02	.45E-06	.10E-05
	1			7		6	−.323486790E+02	.45E-08	.10E-05
p93	0	6	2	15	58	12	.135075968E+03	.30E-03	.10E-02
	1			14	34	14	.135075964E+03	.18E-03	.10E-02
p100	0	7	4	21	102	14	.680630057E+03	.20E-04	.10E-03
	1			18	94	15	.680630057E+03	.86E-04	.10E-03
p110	0	10	0	9	0	8	−.457784697E+02	.77E-07	.10E-05
	1			9		8	−.457784697E+02	.50E-06	.10E-05
p113	0	10	5	12	108	12	.243063768E+02	.13E-03	.10E-02
	1			12	99	12	.243064357E+02	.14E-03	.10E-02
p117	0	15	5	20	219	19	.323486790E+02	.12E-04	.10E-03
	1			18	93	17	.323486790E+02	.52E-05	.10E-03
p118	0	15	0	19	0	19	.664820450E+03	.42E-29	.10E-07
	1			19		19	.664820450E+03	.42E-29	.10E-07

Table 1: Results for Inequality Constrained Problems with CFSQP Version 2.0

prob	B	nparam	nineqn	nf	ncallf	ncallg	iter	objmax	d0norm	eps
bard	0	3	0	15	168	0	8	.50816326E−01	.63E-09	.50E-05
	1				105		7	.50816868E−01	.42E-05	.50E-05
cb2	0	2	0	3	30	0	6	.19522244E+01	.11E-06	.50E-05
	1				18		6	.19522245E+01	.82E-06	.50E-05
cb3	0	2	0	3	15	0	3	.20000015E+01	.75E-06	.50E-05
	1				15		5	.20000000E+01	.94E-09	.50E-05
colv	0	15	0	6	240	0	21	.32348679E+02	.66E-06	.50E-05
	1				102		17	.32348679E+02	.16E-05	.50E-05
dav	0	4	0	20	341	0	12	.11570644E+03	.50E-06	.50E-05
	1				220		11	.11570644E+03	.47E-06	.50E-05
fr	0	2	0	2	32	0	9	.49489521E+01	.17E-06	.50E-05
	1				20		10	.49489521E+01	.16E-06	.50E-05
hett	0	4	0	5	121	0	13	.24593569E−02	.20E-06	.50E-05
	1				74		11	.24593670E−02	.81E-06	.50E-05
rs	0	4	0	4	70	0	9	−.44000000E+02	.19E-06	.50E-05
	1				68		12	−.44000000E+02	.30E-07	.50E-05
wat6	0	6	0	31	608	0	12	.12717343E−01	.19E-05	.50E-05
	1				433		13	.12717091E−01	.18E-08	.50E-05
wat20	0	20	0	31	1953	0	32	.89564416E−07	.14E-05	.50E-05
	1				1023		32	.89713135E−07	.15E-05	.50E-05
wong	0	7	0	5	182	0	19	.68063006E+03	.27E-05	.50E-05
	1				170		26	.68063006E+03	.40E-05	.50E-05
kiwi1	0	5	0	10	157	0	11	.22600162E+02	.37E-06	.11E-05
	1				130		13	.22600162E+02	.60E-06	.60E-06
kiwi4	0	2	0	2	40	0	9	.22204460E−15	.26E-07	.42E-07
	1				23		9	.16254000E−08	.47E-07	.15E-07
mad1	0	2	0	3	24	0	5	−.38965952E+00	.40E-10	.10E-09
	1				18		6	−.38965951E+00	.89E-10	.10E-09
mad2	0	2	0	3	21	0	5	−.33035714E+00	.42E-08	.10E-09
	1				15		5	−.33035714E+00	.21E-07	.10E-09
mad4	0	2	0	3	21	0	7	−.44891078E+00	.85E-08	.10E-09
	1				24		8	−.44891077E+00	.12E-17	.10E-09
mad5	0	2	0	3	31	0	7	−.10000000E+01	.91E-11	.10E-09
	1				21		7	−.99999974E+00	.29E-06	.10E-09
mad6	0	6	0	163	1087	0	6	.11310473E+00	.20E-10	.10E-09
	1				1141		7	.11310473E+00	.16E-09	.10E-09
mad810	0	10	0	18	291	0	10	.38117396E+00	.24E-15	.10E-09
	1				234		13	.38117396E+00	.19E-09	.10E-09
mad830	0	30	0		1371	0	14	.54762050E+00	.17E-07	.10E-09
	1				1044		17	.54762051E+00	.53E-08	.10E-09
mad850	0	50	0	98	3053	0	20	.57927623E+00	.59E-08	.10E-09
	1				1986		20	.57927694E+00	.16E-06	.10E-09
polk1	0	2	0	2	42	0	11	.27182818E+01	.27E-05	.50E-05
	1				22		11	.27182818E+01	.42E-05	.50E-05
polk2	0	10	0	2	191	0	39	.54598150E+02	.35E-07	.50E-05
	1				138		48	.54598150E+02	.28E-07	.50E-05
polk3	0	11	0	10	236	0	17	.37034827E+01	.29E-05	.50E-05
	1				180		17	.37034827E+01	.37E-05	.50E-05
polk4	0	2	0	3	44	0	8	.10087930E−05	.29E-08	.50E-05
	1				24		8	.36460425E+00	.18E-05	.50E-05
p43m	0	4	1	3	67	32	13	−.44000000E+02	.18E-05	.50E-05
	1				54	22	15	−.44000000E+02	.48E-05	.50E-05
p84m	0	5	4	3	17	20	4	−.52803351E+07	.0	.50E-05
	1				9	12	3	−.52803351E+07	.17E-07	.50E-05
p113m	0	10	5	4	108	127	14	.24306209E+02	.39E-05	.50E-05
	1				84	105	14	.24306210E+02	.39E-05	.50E-05
p117m	0	15	3	3	124	144	21	.32348679E+02	.16E-05	.50E-05
	1				54	57	17	.32348679E+02	.26E-04	.50E-05

Table 2: Results for Minimax Problems with CFSQP Version 2.0

prob	B	nparam	ncallf	ncallg	iter	objective	d0norm	epseqn	SCV
p6	0	2	17	31	10	.274055126E−11	.23E-05	.40E-06	.20E-09
	1		63	73	18	.207905131E−14	.10E-06	.40E-06	.13E-07
p7	0	2	57	69	13	−.173205081E+01	.37E-07	.35E-08	.70E-09
	1		26	29	11	−.173205081E+01	.59E-07	.35E-08	.10E-08
p26	0	3	127	188	51	.270576724E−13	.86E-04	.16E-04	.12E-09
	1		38	38	31	.322181111E−13	.90E-04	.16E-04	.43E-07
p27	0	3	147	196	44	.399982677E−01	.11E-03	.10E-02	.43E-04
	1		1732	1735	207	.399916645E−01	.39E-03	.10E-02	.21E-03
p39	0	4	23	93	17	−.100000001E+01	.20E-04	.75E-04	.64E-06
	1		12	26	12	−.100000064E+01	.25E-04	.75E-04	.90E-08
p40	0	4	5	27	5	−.250000002E+01	.14E-05	.85E-04	.96E-08
	1		5	21	5	−.250000822E+01	.48E-05	.85E-04	.43E-05
p42	0	4	9	15	6	.138578644E+02	.77E-06	.45E-05	.51E-09
	1		7	12	7	.138578652E+02	.62E-04	.45E-05	.33E-06
p46	0	5	62	257	26	.224262539E−10	.59E-05	.50E-04	.57E-10
	1		25	73	14	.461984181E−04	.19E-02	.50E-04	.95E-06
p47	0	5	74	568	38	.162241553E−11	.69E-04	.60E-04	.41E-09
	1		30	124	28	.812545812E−12	.62E-04	.60E-04	.14E-07
p56	0	7	31	395	15	−.345600000E+01	.25E-05	.25E-06	.34E-10
	1		14	60	14	−.345600000E+01	.47E-06	.25E-06	.11E-08
p60	0	3	10	22	10	.325682003E−01	.13E-05	.55E-04	.27E-09
	1		9	14	9	.325687946E−01	.96E-04	.55E-04	.55E-04
p61	0	3	18	72	8	−.143646142E+03	.44E-05	.25E-06	.13E-07
	1		17	54	9	−.143646142E+03	.67E-07	.25E-06	.28E-12
p63	0	3	8	17	8	.961715172E+03	.73E-07	.60E-05	.15E-10
	1		6	10	6	.961715172E+03	.16E-04	.60E-05	.65E-07
p71	0	4	9	31	8	.170140173E+02	.24E-07	.70E-05	.35E-11
	1		6	19	6	.170140173E+02	.16E-04	.70E-05	.28E-08
p74	0	4	14	82	14	.512649811E+04	.33E-06	.65E-05	.21E-10
	1		40	120	40	.512649811E+04	.41E-04	.65E-05	.30E-10
p75	0	4	13	75	13	.517441270E+04	.43E-08	.10E-07	.25E-11
	1		27	81	27	.517441270E+04	.79E-07	.10E-07	.39E-07
p77	0	5	15	65	15	.241505133E+00	.38E-04	.35E-04	.68E-07
	1		19	54	18	.241505211E+00	.52E-04	.35E-04	.14E-05
p78	0	5	9	65	9	−.291970041E+01	.26E-07	.15E-05	.45E-10
	1		8	30	8	−.291970041E+01	.33E-04	.15E-05	.10E-08
p79	0	5	7	42	7	.974340336E−01	.39E-05	.15E-03	.41E-07
	1		10	38	10	.974340335E−01	.21E-05	.15E-03	.40E-07
p80	0	5	66	531	20	.539498478E−01	.14E-07	.15E-07	.25E-12
	1		7	21	7	.539498474E−01	.49E-07	.15E-07	.11E-07
p81	0	5	59	468	20	.539498478E−01	.31E-04	.80E-06	.36E-09
	1		8	24	8	.539498419E−01	.23E-04	.80E-06	.17E-06
p99	0	7	111	489	38	−.831079892E+09	.27E+05	.10E-07	.81E-09
	1		130	1356	130	−.831079886E+09	.45E+07	.10E-07	.51E-01
p107	0	9	16	254	14	.505501180E+04	.16E-05	.10E-07	.48E-09
	1		31	223	28	.505501180E+04	.46E-08	.10E-07	.60E-12

Table 3: Results for General Problems with CFSQP Version 2.0

prob	B	nparam	ncsrl	ncsrn	ncallf	ncallg	$\sum_i \lvert\Xi^{g_i}\rvert$	iter	objective	d0norm	$\lvert\Xi^*\rvert$
cw_2	0	2	0	1	6	3073	501	6	.194466012E+00	.26E-09	2
	1				16	8016		16	.194553486E+00	.71E-04	2
cw_3	0	3	0	1	10	8567	501	14	.533468728E+01	.29E-04	2
	1				12	9735		16	.533468729E+01	.95E-04	2
cw_5	0	3	1	0	47	0	501	47	.430118377E+01	.52E-04	2
	1				47	0		47	.430118377E+01	.52E-04	2
cw_6	0	2	0	1	17	8541	501	15	.971588579E+02	.78E-04	1
	1				16	9037		18	.971588525E+02	.20E-07	1
cw_7	0	3	21	0	4	0	441	3	.100000000E+01	.15E-04	21
	1				4	0		3	.100000000E+01	.15E-04	21
hu_1	0	2	1	0	4	0	501	4	.500000000E+00	.39E-16	1
	1				4	0		4	.500000000E+00	.39E-16	1
hu_2	0	2	1	0	2	0	501	2	.680000000E+00	.17E-16	2
	1				2	0		2	.680000000E+00	.17E-16	2
hu_3	0	2	1	0	3	0	501	2	.367037444E+01	.50E-16	2
	1				3	0		2	.367037444E+01	.50E-16	2
hu_4	0	2	0	1	4	2008	501	4	.686291568E+00	.40E-07	1
	1				4	2006		4	.686291501E+00	.63E-12	1
hu_5	0	2	0	1	6	4842	501	8	.986130516E+00	.22E-07	2
	1				8	5358		10	.986145697E+00	.22E-04	2
hu_6	0	2	0	1	10	6298	501	12	−.202500000E+02	.75E-05	1
	1				7	4668		9	−.202498922E+02	.36E-04	1
hu_7	0	2	0	1	24	9957	501	13	.100086194E+01	.14E-05	2
	1				11	6624		11	.100013243E+01	.68E-04	2
hu_9	0	2	0	1	24	21314	501	25	−.125001810E+02	.60E-10	3
	1				18	12084		19	−.125001800E+02	.48E-07	2
hu_10	0	2	0	1	1	1067	501	3	−.812603979E−31	.57E-14	3
	1				1	1067		3	−.812603977E−31	.57E-16	3
hu_11	0	2	0	1	2	1191	501	2	.200000000E+01	.0	501
	1				2	1191		2	.200000000E+01	.0	501
hu_12	0	10	0	1	4	2008	501	4	.818219540E+01	.16E-09	1
	1				3	1505		3	.818219553E+01	.23E-07	1
hz_1	0	2	0	2	10	14903	1002	12	.100001991E+01	.90E-05	3
	1				20	26200		19	.100000470E+01	.70E-05	3
oet_1	0	3	2	0	18	0	1002	18	.538243119E+00	.10E-15	4
	1				18	0		18	.538243119E+00	.10E-15	4
oet_2	0	3	0	2	6	6270	1002	6	.871596961E−01	.62E-07	3
	1				6	6452		6	.871678945E−01	.83E-05	3
oet_3	0	4	2	0	17	0	1002	17	.447592117E−02	.22E-05	5
	1				17	0		17	.447592117E−02	.22E-05	5
oet_4	0	4	0	2	20	26973	1002	22	.431688565E−02	.53E-04	5
	1				10	11115		12	.433432033E−02	.39E-04	5
oet_5	0	5	0	2	25	42450	1002	22	.265069133E−02	.85E-04	4
	1				28	31863		26	.273715778E−02	.89E-04	4
oet_6	0	5	0	2	25	36414	1002	22	.207057768E−02	.24E-05	6
	1				19	23691		20	.213567195E−02	.66E-04	5
oet_7	0	7	0	2	107	116244	1002	54	.733436750E−04	.32E-04	9
	1				134	126693		74	.139612677E−04	.95E-04	7
pt_1	0	2	1	0	15	0	501	15	.236067918E+00	.62E-14	2
	1				15	0		15	.236067918E+00	.62E-14	2
sch_3	0	3	1	0	48	0	501	48	.430118377E+01	.35E-08	2
	1				48	0		48	.430118377E+01	.35E-08	2

Table 4: Results for Sequentially Related Constraint Problems with CFSQP Version 2.0

prob	B	nparam	nfsr	$\sum \lvert\Omega^{f_i}\rvert$	ncallf	iter	objmax	d0norm	$\lvert\Omega^*\rvert$
hz_1m	0	1	1	501	1503	2	.100000000E+01	.22E-14	2
	1				1002	2	.100000000E+01	.44E-14	2
oet_1m	0	2	1	501	5580	10	.538243119E+00	.48E-13	3
	1				6513	13	.538243119E+00	.38E-16	3
oet_2m	0	2	1	501	2022	4	.871610589E−01	.16E-05	3
	1				2505	5	.871636550E−01	.14E-04	3
oet_3m	0	3	1	501	4725	7	.450551698E−02	.23E-05	5
	1				3507	7	.450551698E−02	.23E-05	5
oet_4m	0	3	1	501	6868	10	.429566474E−02	.11E-05	5
	1				5010	10	.429567387E−02	.13E-05	5
oet_5m	0	4	1	501	16869	19	.265008668E−02	.27E-05	4
	1				11522	19	.268822747E−02	.53E-04	4
oet_6m	0	4	1	501	10597	14	.206997910E−02	.17E-04	6
	1				10029	16	.206974500E−02	.21E-07	5
oet_7m	0	6	1	501	26952	30	.132727016E−03	.32E-04	9
	1				84938	97	.463302688E−04	.42E-04	9
pt_1m	0	1	1	501	7337	8	.236067918E+00	.0	2
	1				6012	12	.236067918E+00	.0	2
sch_u3	0	5	1	501	9531	11	.125766196E−03	.11E-04	10
	1				6516	13	.125483347E−03	.10E-05	8

Table 5: Results for Sequentially Related Objective Problems with CFSQP Version 2.0