SLICOT Working Note 2002-3

Definition and Implementation of a SLICOT Interface and a MATLAB Gateway for the Solution of Non-linear Programming Problems [1]

Fernando Alvarruiz[2], Vicente Hernández[2]

March 2002

[1]This document presents research results of the European Community BRITE-EURAM III Thematic Networks Programme NICONET (contract number BRRT-CT97-5040) and is distributed by the Working Group on Software WGS. *WGS secretariat*: Mrs. Ida Tassens, ESAT - Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, 3001-Leuven-Heverlee, BELGIUM. This report is available by anonymous ftp from `wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS/SLWN2002-3.ps.Z`

[2]Departamento de Sistemas Informáticos y Computación (DSIC). Universidad Politécnica de Valencia (UPV). Valencia. Spain. *fbermejo@dsic.upv.es, vhernand@dsic.upv.es.*

Contents

1 Introduction

Non-linear Programming is one of the problems tackled in Task V of the NICONET project (non-linear control problems). A non-linear programming problem is an optimization problem with a non-linear objective function subject to non-linear constraints, given by

$$\text{minimize } f(x)$$

$$\text{subject to} \quad \begin{array}{ll} g_i(x) <= 0, & 1 \le i \le n_i \\ h_i(x) = 0, & 1 \le i \le n_e, \end{array}$$

where $x \in R^n$, and the objective function $f(x)$ and some of the constraint functions $g_i(x)$, $h_i(x)$, are nonlinear.

Following an approach similar to the work done in NICONET for other problems in task V, such as the solution of ordinary differential equations (ODE) and differential algebraic equations (DAE) systems, SLICOT and MATLAB interfaces have been implemented on top of existing software packages for the solution of non-linear programming problems. However, general purpose software libraries for the solution of this problem are not very common. Thus, only one such library, known as FSQP, has been considered.

This paper presents SLICOT and MATLAB interfaces for the FSQP package. The SLICOT interface enables the user to call the FSQP package by means of a subroutine with a SLICOT-compliant calling sequence. By means of the MATLAB interface the user can call the package from MATLAB, defining the problem by means of MATLAB functions.

The interfaces could be extended in the future in order to consider other non-linear programming solvers, although some restructuring of the interfaces would be necessary.

2 The FSQP Package

The FSQP package [2], which stands for *Feasible Sequential Quadratic Programming*, was originally developed by the group of André Tits, at the Institute for Systems Research (ISR), University of Maryland. In year 2000 it was transferred to the company AEM Design (http://gachinese.com/aemdesign).

The package is free for academic institutions, although other research organisations and companies have to pay for its use.

There are two versions of the FSQP package, one implemented in C and the other in Fortran. Because SLICOT has been developed in Fortran, the Fortran version of FSQP has been chosen for the interfaces described here.

The problem to be solved can be described as the minimization of the maximum of a set of smooth objective functions (possibly a single one or none at all) subject to nonlinear equality and inequality constraints, linear equality and inequality constraints, and simple bounds on the variables. Specifically,

$$\text{minimize} \max_{i=1,\ldots,nf} f_i(x),$$

with $x \in R^n$, subject to

$$\begin{array}{ll} bl \leq x \leq bu & \\ g_j(x) \leq 0, & j = 1, \ldots, n_i \\ g_j(x) \equiv x^T c_{j-n_i} - d_{j-n_i} \leq 0, & j = n_i + 1, \ldots, t_i \\ h_j(x) = 0, & j = 1, \ldots, n_e \\ h_j(x) \equiv x^T a_{j-n_e} - b_{j-n_e} = 0, & j = n_e + 1, \ldots, t_e. \end{array}$$

FSQP transforms the original problem in a modified optimization problem with only linear (equality and inequality) constraints and nonlinear inequality constraints. For the transformed problem, it implements algorithms based on a Sequential Quadratic Programming iteration method, modified so as to generate feasible iterates (see [2] for further details).

The FSQP package contains a single user-callable routine, with the following calling sequence

```
SUBROUTINE FFSQP( NPARAM, NF, NINEQN, NINEQ, NEQN, NEQ, MODE, IPRINT, MITER,
                  INFORM, BIGBND, EPS, EPSEQN, UDELTA, BL, BU, X, F, G, IW,
                  IWSIZE, W, WSIZE, OBJ, CONSTR, GRADOB, GRADCN)
```

`NPARAM` is the the dimension of the x vector (n); `NF` is the number of objective functions (nf); `NINEQN`, `NINEQ`, `NEQN` and `NEQ` correspond to the number of the different kinds of constrains (n_i, t_i, n_e and t_e respectively). `X` contains on input the initial vector x, and on output returns the solution of the minimization problem (for a complete description of the arguments, refer to [2]).

Some of the arguments (`OBJ`, `CONSTR`, `GRADOB` and `GRADCN`) correspond to routines that must be defined by the user for each particular problem. These routines are the following:

- `SUBROUTINE OBJ(NPARAM,J,X,FJ)`
  Computes the objective function `J` at `X`, i.e. $f_j(x)$.
- `SUBROUTINE CONSTR(NPARAM,J,X,GJ)`
  Computes the constraint `J` at `X`. If $1 \leq j \leq t_i$, j corresponds to the inequality constraint $g_j(x)$, while if $t_i + 1 \leq j \leq t_i + t_e$, j corresponds to the equality constraint $h_{j-t_i}(x)$.
- `SUBROUTINE GRADOB(NPARAM,J,X,GRADFJ,DUMMY)`
  Computes the gradient of the objective function `J` at `X`, i.e. $\nabla f_j(x) \in R^n$.
- `SUBROUTINE GRADCN (NPARAM,J,X,GRADGJ,DUMMY)`
  Computes the gradient of the constraint function `J` at `X`. If $1 \leq j \leq t_i$, j corresponds to the inequality constraint $g_j(x)$, while if $t_i + 1 \leq j \leq t_i + t_e$, j corresponds to the equality constraint $h_{j-t_i}(x)$.

The package contains two routines, named `GROBFD` and `GRCNFD`, that automatically compute the gradients of the objective functions and the contraint functions, respectively, by means of finite differences. In order to use them, the user must supply these routines as the arguments `GRADOB` and `GRADCN`, respectively.

3 SLICOT Interface

The SLICOT interface for FSQP has followed the standards for the production of SLICOT software [1]. The user-callable routine of this interface presents the following form

```
SUBROUTINE FSQP( MODE, IPRINT, OBJ, CONSTR, GRADOB, GRADCN,
                 NPARAM, NF, NINEQN, NINEQ, NEQN, NEQ, MITER,
                 UDELTA, BIGBND, BL, BU, X, F, G, TOL1, TOL2,
                 IWORK, LIWORK, DWORK, LDWORK, INFO)
```

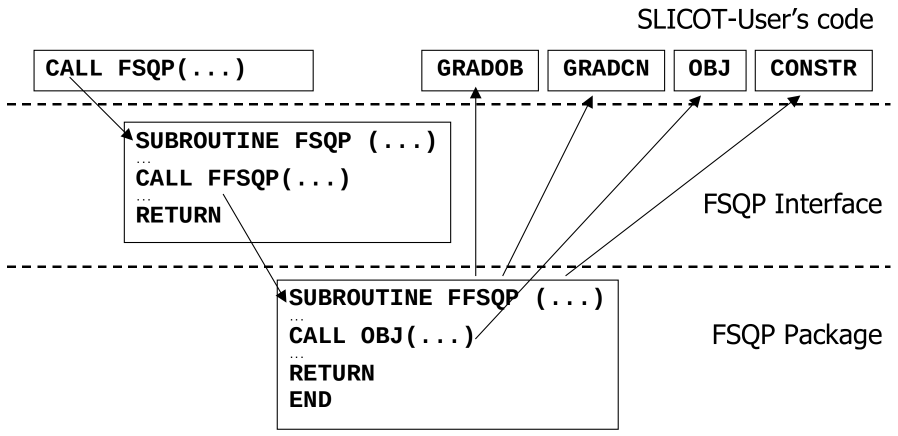

Figure 1: Application using the SLICOT interface.

As can be seen, the SLICOT interface introduces few changes with respect to the original FSQP user interface. Only the order of the arguments and the name of some of them is changed, in order to adapt the calling sequence to the SLICOT standard. In particular, the argument name changes correspond to arguments indicating tolerances, arguments related to work arrays, and the error indicator INFO. A detailed description of the arguments is provided in the source code of the routine (see Appendix A of this Working Note).

The calling sequences of the user-defined subroutines have not been altered in the SLICOT interface. As when using the FSQP package directly, in order to automatically compute the gradients by finite differences, the routines `GROBFD` and `GRCNFD` must be specified for the arguments `GRADOB` and `GRADCN`, respectively.

Figure 1 shows a diagram where we can see the different layers of an application that makes use of the SLICOT interface described here.

4 MATLAB Gateway

The user-callable routine in the MATLAB interface for FSQP presents the following form

```
[x,f,g,lambda,info] = fsqp(mode,iprint,obj,constr,gradob,gradcn,nf,nineqn,
                           nineq,neqn,neq,miter,udelta,bigbnd,bl,bu,x0,tol1,tol2);
```

As can be seen, this routine presents essentially the same form as that of the SLICOT interface. With respect to the user-defined functions, these are similar to those of the FSQP package, with the only changes coming from the adaptation to the MATLAB syntax, as we can see next:

```
function fj = obj(x,j),
```

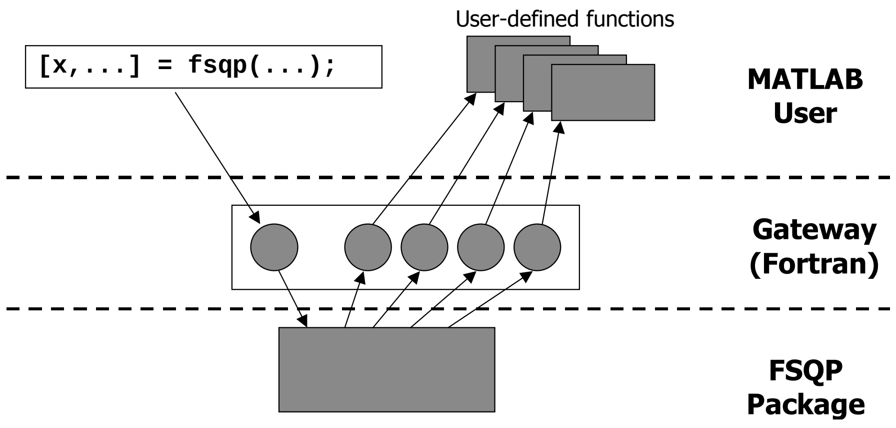

Figure 2: Diagram of the MATLAB interface.

```
function gj = constr(x,j),

function gradfj = gradob (x,j),

function gradgj = gradcn (x,j).
```

For computing gradients by finite differences, use an empty string ('') for the arguments `gradob` and/or `gradcn`. A deatailed description of the `fsqp` routine and the user provided routines is provided in Appendix B of this Working Note.

The MATLAB interface is implemented by means of a gateway written in Fortran. This gateway has an entry point from MATLAB, which calls the FSQP package, and one routine for each of the user-provided routines. These gateway routines only make the call to the corresponding MATLAB user function, retrieving the results. Figure 2 shows a diagram where we can see the different layers involved when the MATLAB interface for FSQP is used.

5 Examples of use

The three examples coming with the FSQP package have been adapted to be solved by means of the SLICOT and MATLAB interfaces. Here we present these examples.

5.1 Example 1

We consider the following problem, taken from [2]

$$\text{minimize } f(x) = (x_1 + 3x_2 + x_3)^2 + 4(x_1 - x_2)^2$$

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

We start from a feasible initial guess $x_0 = (0.1, 0.7, 0.2)^T$. The final solution is $x^\star = (0, 0, 1)^T$ with $f(x^\star) = 1$.

5.1.1 SLICOT interface

A Fortran program using the SLICOT interface for solving the problem could be as follows.

```
      PROGRAM SAMPL1
C
C     Example 1 for SLICOT interface for FSQP
C
C     CONTRIBUTOR
C
C     Fernando Alvarruiz, Vicente Hernandez
C     Universidad Politecnica de Valencia, Spain
C
C
      INTEGER LIWORK,LDWORK,NPARAM,NF,NINEQ,NEQ
      PARAMETER (LIWORK=29, LDWORK=219)
      PARAMETER (NPARAM=3, NF=1)
      PARAMETER (NINEQ=1, NEQ=1)
      INTEGER IWORK(LIWORK)
      DOUBLE  PRECISION X(NPARAM),BL(NPARAM),BU(NPARAM),F(NF+1),
     *        G(NINEQ+NEQ+1),DWORK(LDWORK)
      EXTERNAL OBJ32,CNTR32,GROB32,GRCN32
C
      INTEGER MODE,IPRINT,MITER,NEQN,NINEQN,INFO
      DOUBLE PRECISION BIGBND,TOL1,TOL2,UDELTA
C
      MODE=100
      IPRINT=1
      MITER=500
      BIGBND=1.D+10
      TOL1=1.D-08
      TOL2=0.D0
      UDELTA=0.D0
C
      NEQN=0
      NINEQN=1
C
      BL(1)=0.D0
      BL(2)=0.D0
      BL(3)=0.D0
      BU(1)=BIGBND
      BU(2)=BIGBND
      BU(3)=BIGBND
C
C     Give the initial value of x
C
      X(1)=0.1D0
      X(2)=0.7D0
      X(3)=0.2D0
C
      CALL FSQP( MODE, IPRINT, OBJ32, CNTR32, GROB32, GRCN32,
     &           NPARAM, NF, NINEQN, NINEQ, NEQN, NEQ, MITER,
     &           UDELTA, BIGBND, BL, BU, X, F, G, TOL1, TOL2,
     &           IWORK, LIWORK, DWORK, LDWORK, INFO)
C
      END
```

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

```
      SUBROUTINE OBJ32(NPARAM,J,X,FJ)
      INTEGER NPARAM,J
      DOUBLE PRECISION X(NPARAM),FJ
C
      FJ=(X(1)+3.D0*X(2)+X(3))**2+4.D0*(X(1)-X(2))**2
      RETURN
      END
C
      SUBROUTINE GROB32(NPARAM,J,X,GRADFJ,DUMMY)
      INTEGER NPARAM,J
      DOUBLE PRECISION X(NPARAM),GRADFJ(NPARAM),
     *       FA,FB
      EXTERNAL DUMMY
C
      FA=2.D0*(X(1)+3.D0*X(2)+X(3))
      FB=8.D0*(X(1)-X(2))
      GRADFJ(1)=FA+FB
      GRADFJ(2)=FA*3.D0-FB
      GRADFJ(3)=FA
      RETURN
      END
C
      SUBROUTINE CNTR32(NPARAM,J,X,GJ)
      INTEGER NPARAM,J
      DOUBLE PRECISION X(NPARAM),GJ
C
      GO TO (10,20),J
 10   GJ=X(1)**3-6.0D0*X(2)-4.0D0*X(3)+3.D0
      RETURN
 20   GJ=1.0D0-X(1)-X(2)-X(3)
      RETURN
      END
C
      SUBROUTINE GRCN32(NPARAM,J,X,GRADGJ,DUMMY)
      INTEGER NPARAM,J
      DOUBLE PRECISION X(NPARAM),GRADGJ(NPARAM)
      EXTERNAL DUMMY
C
      GO TO (10,20),J
 10   GRADGJ(1)=3.D0*X(1)**2
      GRADGJ(2)=-6.D0
      GRADGJ(3)=-4.D0
      RETURN
 20   GRADGJ(1)=-1.D0
      GRADGJ(2)=-1.D0
      GRADGJ(3)=-1.D0
      RETURN
      END
```

When running the program (in a Linux PC), the following output is obtained:

```
   FFSQP Version 3.7b  (Released January 1998)
            Copyright (c) 1989 --- 1998
               J.L. Zhou, A.L. Tits,
                 and C.T. Lawrence
                All Rights Reserved


 The given initial point is feasible for inequality
        constraints and linear equality constraints:
 x                      0.10000000000000E+00
                        0.70000000000000E+00
                        0.20000000000000E+00
 objectives             0.72000000000000E+01
 constraints           -0.19990000000000E+01
                        0.27755575615629E-16


 iteration                                   3
 x                       0.00000000000000E+00
                         0.00000000000000E+00
                         0.10000000000000E+01
 objectives              0.10000000000000E+01
 constraints            -0.10000000000000E+01
                         0.00000000000000E+00
 d0norm                  0.32641487870700E-16
 ktnorm                  0.24834349439797E-15
 ncallf                                     3
 ncallg                                     5


 inform                                     0
 Normal termination: You have obtained a solution !!
```

5.1.2 MATLAB interface

To solve the problem from MATLAB, the following code would be adequate.

```
mode=100;
iprint=0;

nf=1;
nineqn=1;
nineq=1;
neqn=0;
neq=1;
miter=100;

bigbnd=1e10;
bl=zeros(3,1);
bu=ones(3,1)*bigbnd;

x0=[0.1 0.7 0.2]';

udelta=0;
tol1=1e-8;
tol2=0;

[x,f,g,lambda,info] = fsqp(mode,iprint,'obj','constr','gradob','gradcn',...
                           nf,nineqn,nineq,neqn,neq,miter,udelta,bigbnd,...
                           bl,bu,x0,tol1,tol2);
```

Following is the definition of the functions `obj`, `constr`, `gradob` and `gradcn` (each function should be defined in a separate file).

```
function fj = obj(x,j)
fj = (x(1)+3*x(2)+x(3))^2 + 4*(x(1)-x(2))^2;

function gj = constr(x,j)
if j==1
    gj = x(1)^3-6*x(2)-4*x(3)+3;
elseif j==2
    gj = 1-x(1)-x(2)-x(3);
end

function gradfj = gradob (x,j)
gradfj = zeros(size(x));
a = 2*(x(1)+3*x(2)+x(3));
b = 8*(x(1)-x(2));
gradfj(1) = a+b;
gradfj(2) = 3*a-b;
gradfj(3) = a;

function gradgj = gradcn (x,j)
gradgj=zeros(size(x));
if j==1
    gradgj(1) = 3*x(1)^2;
    gradgj(2) = -6;
    gradgj(3) = -4;
elseif j==2
    gradgj(1) = -1;
    gradgj(2) = -1;
    gradgj(3) = -1;
end
```

The information returned by the call to the `fsqp` function is the following:

```
x =

  -0.00000000000000
                  0
   1.00000000000000


f =

     1


g =
    -1
     0


lambda =

     0
    -4
     0
     0
     2


info =

     0
```

5.2 Example 2

The second example is the formulated as follows

$$\min_{x\in R^6} \quad \max_{i=1,\ldots,163} |f_i(x)|$$

$$\begin{array}{lrrrrrrrrrcl}
\text{subject to} & - & x_1 & & & & & & & & +s & \le & 0 \\
& & x_1 & - & x_2 & & & & & & +s & \le & 0 \\
& & & & x_2 & - & x_3 & & & & +s & \le & 0 \\
& & & & & & x_3 & - & x_4 & & +s & \le & 0 \\
& & & & & & & & x_4 & - \; x_5 & +s & \le & 0 \\
& & & & & & & & & x_5 - x_6 & +s & \le & 0 \\
& & & & & & & & & x_6 \; -3.5 & +s & \le & 0;
\end{array}$$

where

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

The feasible initial guess is $x_0 = (0.5, 1, 1.5, 2, 2.5, 3)^T$.

5.2.1 SLICOT interface

A suitable main program is as follows.

```
      program sampl2
C
```

```
C     Example 2 for SLICOT interface for FSQP
C
C     CONTRIBUTOR
C
C     Fernando Alvarruiz, Vicente Hernandez
C     Universidad Politecnica de Valencia, Spain
C
C
      integer   ldwork,liwork,nparam,nf,nineq,neq
      parameter (liwork=1029, ldwork=7693)
      parameter (nparam=6, nf=163, nineq=7, neq=0)
      integer   nineqn,neqn,mode,iprint,miter,info,
     *          iwork(liwork)
      double precision bigbnd,tol1,tol2,udelta
      double precision x(nparam),bl(nparam),bu(nparam),dwork(ldwork),
     *                 f(nf+1),g(nineq+neq+1)
      external objmad,cnmad,grobfd,grcnfd
c
      mode=111
      iprint=1
      miter=500
      bigbnd=1.d+10
      tol1=1.d-08
      tol2=0.d0
      udelta=0.d0
c
      nineqn=0
      neqn=0
c
      bl(1)=-bigbnd
      bl(2)=-bigbnd
      bl(3)=-bigbnd
      bl(4)=-bigbnd
      bl(5)=-bigbnd
      bl(6)=-bigbnd
      bu(1)=bigbnd
      bu(2)=bigbnd
      bu(3)=bigbnd
      bu(4)=bigbnd
      bu(5)=bigbnd
      bu(6)=bigbnd
c
c     give the initial value of x
c
      x(1) =  0.5d0
      x(2) =  1.d0
      x(3) =  1.5d0
      x(4) =  2.d0
      x(5) =  2.5d0
      x(6) =  3.d0
c
      CALL FSQP( MODE, IPRINT, OBJMAD, CNMAD, GROBFD, GRCNFD,
     &           NPARAM, NF, NINEQN, NINEQ, NEQN, NEQ, MITER,
     &           UDELTA, BIGBND, BL, BU, X, F, G, TOL1, TOL2,
     &           IWORK, LIWORK, DWORK, LDWORK, INFO)
c
      end
```

In this case we choose to compute the gradients automatically by finite differences. Consequently we need only to provide subroutines for computing the objective and constraint functions, as presented next.

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

```
50   gj=ss+x(4)-x(5)
     return
60   gj=ss+x(5)-x(6)
     return
70   gj=ss+x(6)-3.5d0
     return
     end
```

Running the program on a Linux PC yields the following output:

```
  FFSQP Version 3.7b  (Released January 1998)
           Copyright (c) 1989 --- 1998
              J.L. Zhou, A.L. Tits,
                and C.T. Lawrence
               All Rights Reserved


The given initial point is feasible for inequality
       constraints and linear equality constraints:
x                      0.50000000000000E+00
                       0.10000000000000E+01
                       0.15000000000000E+01
                       0.20000000000000E+01
                       0.25000000000000E+01
                       0.30000000000000E+01
objectives            -0.14191214437885E+00
( ...............           deleted lines        ............... )
                      -0.66666666666667E-01
objmax                 0.22051986506559E+00
constraints           -0.75000000000000E-01
                      -0.75000000000000E-01
                      -0.75000000000000E-01
                      -0.75000000000000E-01
                      -0.75000000000000E-01
                      -0.75000000000000E-01
                      -0.75000000000000E-01


iteration                                 7
x                      0.42500000000000E+00
                       0.85000000000000E+00
                       0.12750000000000E+01
                       0.17000000000000E+01
                       0.21840763196688E+01
                       0.28732755096448E+01
```

```
 objectives           -0.20438068141240E-01
 ( ................         deleted lines       ................ )
                      -0.22240201835557E-01
 objective max4        0.11421841312599E+00
 objmax                0.11310472749827E+00
 constraints           0.00000000000000E+00
                       0.00000000000000E+00
                       0.55511151231258E-16
                      -0.55511151231258E-16
                      -0.59076319668831E-01
                      -0.26419918997598E+00
                      -0.20172449035519E+00
 d0norm                0.15665669556443E-09
 ktnorm                0.20568477115919E-10
 ncallf                               1141


 inform                                  0
 Normal termination: You have obtained a solution !!
```

5.2.2 MATLAB interface

In order to solve the problem from MATLAB, we make use of the following script

```
mode=111;
iprint=0;

nf=163;
nineqn=0;
nineq=7;
neqn=0;
neq=0;
miter=100;

bigbnd=1e10;
bl=-ones(6,1)*bigbnd;
bu=ones(6,1)*bigbnd;

x0=[0.5 1.0 1.5 2.0 2.5 3.0]';

udelta=0;
tol1=1e-8;
tol2=0;

[x,f,g,lambda,info] = fsqp(mode,iprint,'obj','constr','gradob','gradcn',...
                                      nf,nineqn,nineq,neqn,neq,miter,udelta,bigbnd,...
                                      bl,bu,x0,tol1,tol2);
```

The definition of the objective and contraint functions, and their gradients are provided in the following functions. As in the case of the SLICOT interface, we could let FSQP compute gradients by finite differences. However, we choose instead to provide functions for computing them analitically.

```
function fj = obj(x,j)
theta = pi/180 * (8.5+0.5*j);
fj=0;
for i=1:6
    fj = fj + cos(2*pi*x(i)*sin(theta));
end
fj = fj + cos(7*pi*sin(theta));
fj = 1/15 + 2/15*fj;


function gj = constr(x,j)
if j==1
    a=0;
else
    a=x(j-1);
end
if j==7
    b=3.5;
else
    b=x(j);
end
gj = a - b + 0.425;


function gradfj = gradob (x,j)
theta = pi/180 * (8.5+0.5*j);
gradfj = -2/15 * sin(2*pi*x*sin(theta)) * 2*pi*sin(theta);


function gradgj = gradcn (x,j)
gradgj=zeros(size(x));
if j>1
    gradgj(j-1) = 1;
end
if j<7
    gradgj(j) = -1;
end
```

The solution given by the call to the `fsqp` function (on a Windows PC with Matlab 6) is the following:

```
x =

   0.42500000000000
   0.85000000000000
   1.27500000000000
   1.70000000000000
   2.18407631966944
   2.87327550964551
```

5.3 Example 3

The third example is defined by

$$\min_{x \in R^4} x_1 x_4 (x_1 + x_2 + x_3) + x_3$$

$$\begin{aligned} \text{subject to} \quad & 1 \le x_i \le 5, i = 1, \ldots, 4 \\ & x_1 x_2 x_3 x_4 - 25 \le 0 \\ & x_1^2 + x_2^2 + x_3^2 + x_4^2 - 40 = 0 \end{aligned}$$

We start from the feasible initial guess $x_0 = (1.5, 5, 1)^T$.

5.3.1 SLICOT interface

The following Fortran program is adequate for the solution of the problem

```
      program sampl3
C
C     Example 3 for SLICOT interface for FSQP
C
C     CONTRIBUTOR
C
C     Fernando Alvarruiz, Vicente Hernandez
C     Universidad Politecnica de Valencia, Spain
C
C
      integer liwork,ldwork,nparam,nf,nineq,neq
      parameter (liwork=33, ldwork=284)
      parameter (nparam=4, nf=1)
      parameter (nineq=1, neq=1)
      integer iwork(liwork)
      double  precision x(nparam),bl(nparam),bu(nparam),f(nf+1),
     *        g(nineq+neq+1),dwork(ldwork)
```

```
      external obj,constr,gradob,gradcn
c
      integer mode,iprint,miter,neqn,nineqn,info
      double precision bigbnd,tol1,tol2,udelta
c
      mode=100
      iprint=1
      miter=500
      bigbnd=1.d+10
      tol1=1.d-07
      tol2=7.d-6
      udelta=0.d0
c
      neqn=1
      nineqn=1
      bl(1)=1.d0
      bl(2)=1.d0
      bl(3)=1.d0
      bl(4)=1.d0
      bu(1)=5.d0
      bu(2)=5.d0
      bu(3)=5.d0
      bu(4)=5.d0
c
c     give the initial value of x
c
      x(1)=1.d0
      x(2)=5.d0
      x(3)=5.d0
      x(4)=1.d0
c
      CALL FSQP( MODE, IPRINT, OBJ, CONSTR, GRADOB, GRADCN,
     &                 NPARAM, NF, NINEQN, NINEQ, NEQN, NEQ, MITER,
     &                 UDELTA, BIGBND, BL, BU, X, F, G, TOL1, TOL2,
     &                 IWORK, LIWORK, DWORK, LDWORK, INFO)
c
      end
```

The following subroutines define the objective and constraint functions, and their gradients.

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

The output produced by the program (on a Linux PC) is presented next.

```
   FFSQP Version 3.7b  (Released January 1998)
            Copyright (c) 1989 --- 1998
               J.L. Zhou, A.L. Tits,
                 and C.T. Lawrence
                All Rights Reserved


 The given initial point is feasible for inequality
        constraints and linear equality constraints:
 x                      0.10000000000000E+01
                        0.50000000000000E+01
                        0.50000000000000E+01
                        0.10000000000000E+01
 objectives             0.16000000000000E+02
 constraints            0.00000000000000E+00
                       -0.12000000000000E+02


 iteration                                 7
 x                      0.10000000000000E+01
                        0.47429996304683E+01
                        0.38211499930642E+01
                        0.13794082919438E+01
 objectives             0.17014017289156E+02
 constraints           -0.40611958240788E-12
                       -0.40293463010599E-12
 SNECV                  0.40293463010599E-12
 d0norm                 0.11204974331019E-07
 ktnorm                 0.19065820825416E-07
 ncallf                                    7
 ncallg                                   30


 inform                                    0
 Normal termination: You have obtained a solution !!
```

5.3.2 MATLAB interface

The following script is used in order to solve the problem by means of the MATLAB interface to FSQP.

```
mode=100;
iprint=0;

nparam=4;
```

```
nf=1;
nineqn=1;
nineq=1;
neqn=1;
neq=1;
miter=100;

bigbnd=1e10;
bl=ones(nparam,1);
bu=ones(nparam,1)*5;

x0=[1,5,5,1]';

udelta=0;
tol1=1e-7;
tol2=7e-6;

[x,f,g,lambda,info] = fsqp(mode,iprint,'obj','constr','gradob','gradcn',...
                      nf,nineqn,nineq,neqn,neq,miter,udelta,bigbnd,bl,bu,...
                      x0,tol1,tol2)
```

The definition of the objective and constraint functions and their gradients is shown next.

```
function fj = obj(x,j)
fj = x(1)*x(4)*(x(1)+x(2)+x(3)) + x(3);


function gj = constr(x,j)
if j==1
    gj = -x(1)*x(2)*x(3)*x(4)+25;
elseif j==2
    gj = x(1)^2 + x(2)^2 + x(3)^2 + x(4)^2 -40;
end


function gradfj = gradob (x,j)
gradfj = zeros(4,1);
gradfj(1) = (2*x(1)+x(2)+x(3))*x(4);
gradfj(2) = x(1)*x(4);
gradfj(3) = x(1)*x(4) + 1;
gradfj(4) = x(1)*(x(1)+x(2)+x(3));


function gradgj = gradcn (x,j)
gradgj = zeros(4,1);
```

```
if j==1
    gradgj(1) = -x(2)*x(3)*x(4);
    gradgj(2) = -x(1)*x(3)*x(4);
    gradgj(3) = -x(1)*x(2)*x(4);
    gradgj(4) = -x(1)*x(2)*x(3);
elseif j==2
    gradgj = 2*x;
end
```

And finally, the results obtained by the call to function `fsqp` (on a Windows PC with Matlab 6) are as shown below.

```
x =

   1.00000000000000
   4.74299963046833
   3.82114999306416
   1.37940829194384


f =

  17.01401728915646


g =

  1.0e-012 *

  -0.40500935938326
  -0.40500935938326


lambda =

  -1.08787122822622
                  0
                  0
                  0
   0.55229366019751
   0.16146856677014
```

6 Caveats

FSQP contains some named `COMMON` blocks. In particular, the following block

```
      DOUBLE  PRECISION OBJEPS,OBJREP,GLGEPS
      LOGICAL XISNEW
      COMMON  /FSQPUS/OBJEPS,OBJREP,GLGEPS,XISNEW
```

defines variables that can be accessed and modified by the user, affecting the behaviour of the call to the user-callable routine. The variables `OBJEPS`, `OBJREP` and `GLGEPS` can be used to define alternative stopping criteria. The logical variable `XISNEW` is initially set to `.TRUE.` and reset to `.TRUE.` whenever FSQP changes the value of the current iterate x. The user may test this variable in the user-defined subroutines in order to do all function evaluations at once, when x is first changed, and then set `XISNEW` to `.FALSE.`.

Of course, this `COMMON` block is accessible from a Fortran program even if the SLICOT interface for FSQP is used. However, the current implementation of the MATLAB interface does not consider this `COMMON` block a crucial feature, thus no mechanism for interfacing with it is provided. In any case, this could be added in a later version if convenient.

References

[1] Working Group on Software (WGS), *SLICOT Implementation and Documentation Standards*, WGS-Report 96-1, Eindhoven University of Technology, February 1998, ftp://wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS/rep96-1.ps.Z.

[2] Jian L. Zhou, André L. Tits, and Craig T. Lawrence *User's Guide for FFSQP Version 3.7: A Fortran Code for Solving Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality and Linear Constraints*, University of Maryland, College Park, MD 20742.

A Documentation of the SLICOT interface for FSQP

We present here the source code of the routine `FSQP`, which implements the SLICOT interface for FSQP.

```
      SUBROUTINE FSQP( MODE, IPRINT, OBJ, CONSTR, GRADOB, GRADCN,
     &                 NPARAM, NF, NINEQN, NINEQ, NEQN, NEQ, MITER,
     &                 UDELTA, BIGBND, BL, BU, X, F, G, TOL1, TOL2,
     &                 IWORK, LIWORK, DWORK, LDWORK, INFO)
C
C     WGS COPYRIGHT 2000.
C
C     PURPOSE
C
C     To perform the minimization of the maximum of a set of smooth
C     objective functions (possibly a single one or none at all) subject
C     to nonlinear equality and inequality constraints, linear equality
```

```
C     and inequality constraints, and simple bounds on the variables.
C     Specifically, the problem to solve is of the form
C
C                                 min max{f (x)}
C                                      i   i
C
C                                                     n
C     where 1 <= i <= NF, and x is a vector in R  subject to the
C     following constraints
C
C      bl <= x <= bu
C
C      g (x)<=0,    j = 1 ... NINEQN
C       j
C      g (x)<=0,    j = NINEQN+1 ... NINEQ
C       j
C      h (x)=0,     j = 1 ... NEQN
C       j
C      h (x)=0,     j = NEQN+1 ... NEQ
C       j
C
C                                           n
C     where bl and bu are vectors in R , and
C
C         n
C     f : R -> R,   j = 1 ... NF,           smooth
C      j
C         n
C     g : R -> R,   j = 1 ... NINEQN,       nonlinear and smooth
C      j
C         n
C     g : R -> R,   j = NINEQN+1 ... NINEQ, linear
C      j
C         n
C     h : R -> R,   j = 1 ... NEQN,         nonlinear and smooth
C      j
C         n
C     h : R -> R,   j = NEQN+1 ... NEQ,     linear
C      j
C
C     ARGUMENTS
C
C     Mode Parameters
C
C     MODE    (input) INTEGER
```

```
C           mode = CBA (a 3-digit number) with the following meaning
C           (see [1] for more details):
C
C           A specifies the problem to be solved:
C           A = 0: The problem is that described above.
C           A = 1: In the problem described above, the function to
C                  minimize is replaced by
C                            min max{ABS(f (x))}
C                                 i       i
C                  i.e., absolute values of the objective functions
C                  are taken
C
C           B indicates the method to be used:
C           B = 0: Algorithm FFSQP-AL is selected (see METHOD below).
C           B = 1: Algorithm FFSQP-NL is selected (see METHOD below).
C
C           C indicates the order of evaluation of objective
C           functions and constraints during the line search:
C           C = 1: The function that caused the previous value of t
C                  to be rejected is checked first and all functions
C                  of the same type ("objective" or "constraint") as
C                  the latter will then be checked first (recommended
C                  for most users).
C           C = 2: Constraints will be always checked first at each
C                  trial point during the line search. If it is a
C                  contraint that caused the previous value of t to
C                  be rejected, that constraint will be checked first
C                  (useful when objective functions are not defined or
C                  are difficult to evaluate outside of the feasible
C                  region).
C
C   IPRINT  (input) INTEGER
C           Parameter indicating the desired output (see [1] for a
C           more complete description of the output).
C           IPRINT = 0: No information except for user-input errors
C              is displayed. This value is imposed during phase 1.
C           IPRINT = 1: Objective and constraint values at the
C              initial feasible point are displayed. At the end of
C              execution, status (INFO), iterate, objective values,
C              constraint values, number of evaluations of objectives
C              and nonlinear constraints, norm of the Kuhn-Tucker
C              vector, and sum of feasibility violation are
C              displayed.
C           IPRINT = 2: At the end of each iteration, the same
C              information as with IPRINT = 1 is displayed.
```

```
C           IPRINT = 3: At each iteration, the same information as
C              with IPRINT = 2,including detailed information on the
C              search direction computation, on the line search, and
C              on the update, is displayed.
C           IPRINT = 10*N +M: N any positive integer, M=2 or 3.
C              Information corresponding to IPRINT=M is displayed at
C              every (10*N)th iteration and at the last iteration.
C
C     User-supplied subroutines
C
C     OBJ     SUBROUTINE
C             Computes the value of the objective functions. If NF = 0,
C             a (dummy) subroutine must be provided anyway. The
C             specification of OBJ is
C
C             SUBROUTINE OBJ(NPARAM,J,X,FJ)
C             INTEGER NPARAM, J
C             DOUBLE PRECISION X(NPARAM),FJ
C
C             Arguments:
C             NPARAM (Input) Dimension of X.
C             J (Input) Number of the objective to be computed.
C             X (Input) Current iterate.
C             FJ (Output) Value of the jth objective function at X.
C
C     CONSTR  SUBROUTINE
C             Computes the value of the constraints. If there are no
C             constraints, a (dummy) subroutine must be provided anyway.
C             The specification of CONSTR is as follows.
C
C             SUBROUTINE CONSTR(NPARAM,J,X,GJ)
C             INTEGER NPARAM,J
C             DOUBLE PRECISION X(NPARAM),GJ
C
C             Arguments:
C             NPARAM (Input) Dimension of X.
C             J (Input) Number of the constraint to be computed.
C             X (Input) Current iterate.
C             GJ (Output) Value of the jth constraint at X.
C
C             The order of the constraints must be as follows. First
C             the NINEQN (possibly zero) nonlinear inequality
C             constraints. Then the NINEQ-NINEQN (possibly zero) linear
C             inequality constraints. Finally, the NEQN (possibly zero)
C             nonlinear equality constraints followed by the NEQ-NEQN
```

```
C            (possibly zero) linear equality constraints.
C
C     GRADOB  SUBROUTINE
C             Computes the gradients of the objective functions. The
C             user must pass the subroutine name GROBFD, if he/she
C             wishes that FSQP evaluate these gradients automatically,
C             by forward finite differences. The specification of GRADOB
C             is as follows.
C
C             SUBROUTINE GRADOB(NPARAM,J,X,GRADFJ,DUMMY)
C             INTEGER NPARAM,J
C             DOUBLE PRECISION X(NPARAM),GRADFJ(NPARAM)
C             DOUBLE PRECISION DUMMY
C             EXTERNAL DUMMY
C
C             Arguments:
C             NPARAM (Input) Dimension of X.
C             J (Input) Number of objective for which gradient is to be
C                   computed.
C             X (Input) Current iterate.
C             GRADFJ (Output) Gradient of the jth objective function at
C                   X.
C             DUMMY (Input) Used by GROBFD (internally assigned the
C                   name of the objective function subroutine by FFSQP).
C
C             Note that DUMMY is passed as argument to GRADOB to allow
C             for forward finite difference computation of the gradient.
C
C     GRADCN  SUBROUTINE
C             Computes the gradients of the constraints. The
C             user must pass the subroutine name GRCNFD, if he/she
C             wishes that FSQP evaluate these gradients automatically,
C             by forward finite differences. The specification of GRADCN
C             is as follows
C
C             SUBROUTINE GRADCN (NPARAM,J,X,GRADGJ,DUMMY)
C             INTEGER NPARAM,J
C             DOUBLE PRECISION X(NPARAM),GRADGJ(NPARAM)
C             DOUBLE PRECISION DUMMY
C             EXTERNAL DUMMY
C
C             Arguments:
C             NPARAM (Input) Dimension of X.
C             J (Input) Number of constraint for which gradient is to
C                   be computed.
```

```
C           X (Input) Current iterate.
C           GRADGJ (Output) Gradient of the jth constraint evaluated
C                 at X.
C           DUMMY (Input) Used by GRCNFD (internally assigned the
C                 name of the constraint function subroutine by
C                 FFSQP).
C
C           Note that DUMMY is passed as argument to GRADCN to allow
C           for forward finite difference computation of the gradients.
C
C     Input/Output Parameters
C
C     NPARAM  (input) INTEGER
C             Number of free variables, i.e., the dimension of X.
C
C     NF      (input) INTEGER
C             Number of objective functions (possibly zero).
C
C     NINEQN  (input) INTEGER
C             Number (possibly zero) of nonlinear inequality
C             constraints.
C
C     NINEQ   (input) INTEGER
C             Total number (possibly equal to nineqn) of inequality
C             constraints.
C
C     NEQN    (input) INTEGER
C             Number (possibly zero) of nonlinear equality constraints.
C
C     NEQ     (input) INTEGER
C             Total number (possibly equal to neqn) of equality
C             constraints.
C
C     MITER   (input) INTEGER
C             Maximum number of iterations allowed by the user before
C             termination of execution.
C
C     UDELTA  (input) DOUBLE PRECISION
C             The perturbation size the user suggests to use in
C             approximating gradients by finite difference. UDELTA
C             should be set to zero if the user has no idea how to
C             choose it. See [1] for details.
C
C     BIGBND  (input) DOUBLE PRECISION
C             It plays the role of Infinite Bound (see also BL and BU
C            below).
C
C     BL      (input) DOUBLE PRECISION array, dimension (NPARAM)
C             Lower bounds for the components of X. To specify a non-
C             existent lower bound for some j, the value used must
C             satisfy BL(j) <= -BIGBND.
C
C     BU      (input) DOUBLE PRECISION array, dimension (NPARAM)
C             Upper bounds for the components of X. To specify a non-
C             existent upper bound for some j, the value used must
C             satisfy BU(j) >= BIGBND.
C
C     X       (input/output) DOUBLE PRECISION array, dimension (NPARAM)
C             On entry, this is the initial guess.
C             On exit, this is the iterate at the end of execution.
C
C     F       (output) DOUBLE PRECISION array, dimension ( MAX(1,NF) )
C             Value of functions f_i, i = 1, ..., NF, at X at the end
C             of execution.
C
C     G       (output) DOUBLE PRECISION array, dimension
C             ( MAX(1,NINEQ+NEQ) )
C             Value of constraint functions at X at the end of
C             execution.
C
C     Tolerances
C
C     TOL1    DOUBLE PRECISION
C             Corresponds to argument EPS in [1].
C             Final norm requirement for the Newton direction (see [1],
C             argument EPS). It must be bigger than the machine
C             precision epsmac (computed by FSQP). If the user does
C             not have a good feeling of what value should be chosen,
C             a very small number could be provided and IPRINT = 2 be
C             selected so that the user would be able to keep track of
C             the process of optimization and terminate FSQP at
C             appropriate time.
C
C     TOL2    DOUBLE PRECISION
C             Corresponds to argument EPSEQN in [1].
C             Maximum violation of nonlinear equality constraints
C             allowed by the user at an optimal point. It is in effect
C             only if NEQN > 0 and must be bigger than the machine
C             precision epsmac (computed by FSQP).
C
```

```
C     Workspace
C
C     IWORK   INTEGER array, dimension (LIWORK)
C             Corresponds to argument IW in [1].
C
C     LIWORK  INTEGER
C             Corresponds to argument IWSIZE in [1].
C             The length of array IWORK. It must be at least as big as
C             6*NPARAM + 8*max(1,NINEQ+NEQ) + 7*max(1,NF) + 30. This
C             estimate is usually very conservative and the smallest
C             suitable value will be displayed if the user-supplied
C             value is too small.
C
C     DWORK   DOUBLE PRECISION array, dimension (LDWORK)
C             Corresponds to argument W in [1].
C             On exit, it will contain estimates of Lagrange
C             multipliers at the end of execution. These multipliers
C             will be placed in the first NPARAM + NINEQ + NEQ + NFF
C             entries; where NFF = 0 if (in mode) A = 0 and NF = 1, and
C             NFF = NF otherwise. See [1] for details.
C
C     LDWORK  INTEGER
C             Corresponds to argument NWSIZE in [1].
C             The length of array DWORK. It must be at least as big as
C             4*SQR(NPARAM) + 5*MAX(1,NINEQ+NEQ)*NPARAM +
C             3*MAX(1,NF)*NPARAM + 26*(NPARAM+MAX(1,NF)) +
C             45*MAX(1,NINEQ+NEQ) + 100.
C             This estimate is usually very conservative and the
C             smallest suitable value will be displayed if the user-
C             supplied value is too small.
C
C     Error Indicator
C
C     INFO    (output) INTEGER
C             Corresponds to argument INFORM in [1].
C             INFO = 0: Normal termination of execution.
C             INFO = 1: The user-provided initial guess is infeasible
C                for linear constraints and FFSQP is unable to generate
C                a point satisfying all these constraints.
C             INFO = 2: The user-provided initial guess is infeasible
C                for nonlinear inequality constraints and linear
C                constraints; and FFSQP is unable to generate a point
C                satisfying all these constraints.
C             INFO = 3: The maximum number miter of iterations has
C                been reached before a solution is obtained.
```

```
C            INFO = 4: The line search fails to find a new iterate
C               (trial step size being smaller than the machine
C               precision epsmac computed by FFSQP).
C            INFO = 5: Failure of the QP solver in attempting to
C               construct d0. A more robust QP solver may succeed.
C            INFO = 6: Failure of the QP solver in attempting to
C               construct d1 . A more robust QP solver may succeed.
C            INFO = 7: Input data are not consistent (with printout
C               indicating the error).
C            INFO = 8: Two consecutive iterates are numerically
C               equivalent before a stopping criterion is satisfied.
C            INFO = 9: One of the penalty parameters exceeded
C               BIGBND. The algorithm is having trouble satisfying a
C               nonlinear equality constraint.
C
C     METHOD
C
C     If the initial guess provided by the user is infeasible for
C     nonlinear inequality constraints and linear constraints, FFSQP
C     first generates a point satisfying all these constraints by
C     iterating on the problem of minimizing the maximum of these
C     constraints.
C
C     Then, using Mayne-Polak's scheme, nonlinear equality
C     constraints are turned into nonlinear inequality constraints
C     and the original objective function is replaced by a modified
C     objective function.
C
C     After obtaining feasibility, either (i) an Armijo-type line
C     search may be used (algorithm FFSQP-AL), yielding a monotone
C     decrease of the objective function at each iteration; or (ii) a
C     nonmonotone line search may be selected (algorithm FFSQP-NL),
C     forcing a decrease of the objective function within at most four
C     iterations.
C
C     See [1] for further details.
C
C     REFERENCES
C
C     [1] J. L. Zhou, A. L. Tits and C. T. Lawrence
C         User's Guide for FFSQP Version 3.7 : A Fortran Code for
C         Solving Optimization Programs, Possibly Minimax, with General
C         Inequality Constraints and Linear Equality Constraints,
C         Generating Feasible Iterates
C         Institute for Systems Research, University of Maryland,
C         Technical Report SRC-TR-92-107r5, 1997.
C         (Available at http://gachinese.com/aemdesign/FSQPframe.htm)
C
C     NUMERICAL ASPECTS
C
C     CONTRIBUTOR
C
C     Fernando Alvarruiz, Vicente Hernandez
C     Universidad Politecnica de Valencia, Spain
C
C     REVISIONS
C
C     KEYWORDS
C
C     Sequential Quadratic Programming, Non-Linear Programming
C
C     ******************************************************************
C
      IMPLICIT NONE
C     .. Parameters ..
C     .. Scalar Arguments ..
      INTEGER           NPARAM, NF, NINEQN, NINEQ, NEQN, NEQ, MODE,
     &                  IPRINT, MITER, INFO, LIWORK, LDWORK
      DOUBLE PRECISION  BIGBND, TOL1, TOL2, UDELTA
C     .. Array Arguments ..
      INTEGER           IWORK(LIWORK)
      DOUBLE PRECISION  BL(*), BU(*), X(*), F(*), G(*), DWORK(LDWORK)
C     .. Local Scalars ..
C     .. External Functions ..
C     .. External Subroutines ..
      EXTERNAL          OBJ, CONSTR, GRADOB, GRADCN
      EXTERNAL          FFSQP
C     .. Intrinsic Functions ..
C     .. Executable Statements ..
C
      CALL FFSQP( NPARAM, NF, NINEQN, NINEQ, NEQN, NEQ, MODE, IPRINT,
     &            MITER, INFO, BIGBND, TOL1, TOL2, UDELTA, BL, BU,
     &            X, F, G, IWORK, LIWORK, DWORK, LDWORK, OBJ, CONSTR,
     &            GRADOB, GRADCN)
C
      RETURN
C *** Last line of FSQP ***
      END
```

B Documentation of the MATLAB interface for FSQP

We present here the file `fsqp.m`, which contains only documentation about the use of the MATLAB interface for FSQP. This documentation is displayed when the user types "`help fsqp`" at the MATLAB prompt. The implementation of the MATLAB interface is contained in the Fortran file `fsqp_gtw.f`.

```
% fsqp - Feasible Sequential Quadratic Programming Optimization
%
% [x,f,g,lambda,info] = fsqp(mode,iprint,obj,constr,gradob,gradcn,nf,nineqn,
%                      nineq,neqn,neq,miter,udelta,bigbnd,bl,bu,x0,tol1,tol2);
%
%     PURPOSE
%
%     To perform the minimization of the maximum of a set of smooth
%     objective functions (possibly a single one or none at all) subject
%     to nonlinear equality and inequality constraints, linear equality
%     and inequality constraints, and simple bounds on the variables.
%     Specifically, the problem to solve is of the form
%
%                              min max{f (x)}
%                                   i   i
%
%                                                   n
%     where 1 <= i <= nf, and x is a vector in R  subject to the
%     following constraints
%
%      bl <= x <= bu
%
%      g (x)<=0,    j = 1 ... nineqn
%       j
%      g (x)<=0,    j = nineqn+1 ... nineq
%       j
%      h (x)=0,     j = 1 ... neqn
%       j
%      h (x)=0,     j = neqn+1 ... neq
%       j
%
%                                         n
%     where bl and bu are vectors in R , and
%
%
%         n
%     f : R -> R,   j = 1 ... nf,           smooth
%      j
%         n
%     g : R -> R,   j = 1 ... nineqn,       nonlinear and smooth
%      j
%         n
%     g : R -> R,   j = nineqn+1 ... nineq, linear
%      j
%         n
%     h : R -> R,   j = 1 ... neqn,         nonlinear and smooth
%      j
%         n
%     h : R -> R,   j = neqn+1 ... neq,     linear
%      j
%
%     INPUT ARGUMENTS
%
%     mode    Integer
%             mode = CBA (a 3-digit number) with the following meaning
%             (see [1] for more details):
%
%             A specifies the problem to be solved:
%             A = 0: The problem is that described above.
%             A = 1: In the problem described above, the function to
%                    minimize is replaced by
%                              min max{ABS(f (x))}
%                                   i       i
%                    i.e., absolute values of the objective functions
%                    are taken.
%
%             B indicates the method to be used:
%             B = 0: Algorithm FFSQP-AL is selected (see METHOD below).
%             B = 1: Algorithm FFSQP-NL is selected (see METHOD below).
%
%             C indicates the order of evaluation of objective
%             functions and constraints during the line search:
%             C = 1: 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).
%             C = 2: Constraints will be always checked first at each
%                    trial point during the line search. If it is a
%                    contraint 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).
```

```
%
%     iprint  Integer
%             Parameter indicating the desired output (see [1] for a
%             more complete description of the output).
%             iprint = 0: No information except for 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 (info), iterate, objective values,
%                constraint values, number of evaluations of objectives
%                and nonlinear constraints, norm of the Kuhn-Tucker
%                vector, and sum of feasibility violation are
%                displayed.
%             iprint = 2: At the end of each iteration, the same
%                information as with iprint = 1 is displayed.
%             iprint = 3: At each iteration, the same information as
%                with iprint = 2,including detailed information on the
%                search direction computation, on the line search, and
%                on the update, is displayed.
%             iprint = 10*N +M: N any positive integer, M=2 or 3.
%                Information corresponding to iprint=M is displayed at
%                every (10*N)th iteration and at the last iteration.
%
%     obj     String
%             Name of a matlab .m file that computes the value of the
%             objective functions. The .m file should have a function
%             with the following form
%
%             function fj = obj(x,j)
%
%             where j is the number of the objective function to be
%             evaluated at the point x, and fj is a scalar real number.
%
%     constr  String
%             Name of a matlab .m file that computes the value of the
%             constraint functions. The .m file should have a function
%             with the following form
%
%             function gj = constr(x,j)
%
%             where j is the number of the constraint to be evaluated at
%             point x, and fj is a scalar real number.
%
%             The order of the constraints must be as follows. First
%             the nineqn (possibly zero) nonlinear in-equality
%            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.
%
%    gradob  String
%            Name of a matlab .m file that computes the gradients of the
%            objective functions. The .m file should have a function
%            with the following form
%
%            function gradfj = gradob (x,j)
%
%            where j is the number of the objective function which gradient
%            is to be computed at point x. The result returned, gradfj,
%            must be a (row or column) vector with length equal to length(x).
%            If gradob='', the gradients will be computed automatically
%            by forward finite differences.
%
%    gradcn  String
%            Name of a matlab .m file that computes the gradients of the
%            constraint functions. The .m file should have a function
%            with the following form
%
%            function gradgj = gradcn (x,j)
%
%            where j is the number of the constraint function which gradient
%            is to be computed at point x. The result returned, gradgj,
%            must be a (row or column) vector with length equal to length(x).
%            If gradcn='', the gradients will be computed automatically
%            by forward finite differences.
%
%    nf      Integer
%            Number of objective functions (possibly zero).
%
%    nineqn  Integer
%            Number (possibly zero) of nonlinear inequality
%            constraints.
%
%    nineq   Integer
%            Total number (possibly equal to nineqn) of inequality
%            constraints.
%
%    neqn    Integer
%            Number (possibly zero) of nonlinear equality constraints.
%
```

```
%     neq     Integer
%             Total number (possibly equal to neqn) of equality
%             constraints.
%
%     miter   Integer
%             Maximum number of iterations allowed by the user before
%             termination of execution.
%
%     udelta  Real
%             The perturbation size the user suggests to use in
%             approximating gradients by finite difference. udelta
%             should be set to zero if the user has no idea how to
%             choose it. See [1] for details.
%
%     bigbnd  Real
%             It plays the role of Infinite Bound (see also bl and bu
%             below).
%
%     bl      Real vector
%             Lower bounds for the components of x. To specify a non-
%             existent lower bound for some j, the value used must
%             satisfy bl(j) <= -bigbnd.
%
%     bu      Real vector
%             Upper bounds for the components of x. To specify a non-
%             existent upper bound for some j, the value used must
%             satisfy bu(j) >= bigbnd.
%
%     x0      Real vector
%             Initial guess.
%
%     tol1    Real
%             Corresponds to argument EPS in [1].
%             Final norm requirement for the Newton direction (see [1],
%             argument EPS). It must be bigger than the machine
%             precision epsmac (computed by FSQP). 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 FSQP at
%             appropriate time.
%
%     tol2    Real
%             Corresponds to argument EPSEQN in [1].
%             Maximum violation of nonlinear equality constraints
%            allowed by the user at an optimal point. It is in effect
%            only if neqn > 0 and must be bigger than the machine
%            precision epsmac (computed by FSQP).
%
%     OUTPUT RESULTS
%
%     x      Real vector
%            Solution to the optimization problem.
%
%     f      Real vector
%            Value of functions f_i, i = 1, ..., nf, at x at the end
%            of execution.
%
%     g      Real vector
%            Value of constraint functions at x at the end of
%            execution.
%
%     lambda Real vector
%            Estimates of Lagrange multipliers at the end of execution.
%            See [1] for details.
%
%     info   Integer
%            Corresponds to argument INFORM in [1].
%            info = 0: Normal termination of execution.
%            info = 1: The user-provided initial guess is infeasible
%               for linear constraints and FFSQP is unable to generate
%               a point satisfying all these constraints.
%            info = 2: The user-provided initial guess is infeasible
%               for nonlinear inequality constraints and linear
%               constraints; and FFSQP is unable to generate a point
%               satisfying all these constraints.
%            info = 3: The maximum number miter of iterations has
%               been reached before a solution is obtained.
%            info = 4: The line search fails to find a new iterate
%               (trial step size being smaller than the machine
%               precision epsmac computed by FFSQP).
%            info = 5: Failure of the QP solver in attempting to
%               construct d0. A more robust QP solver may succeed.
%            info = 6: Failure of the QP solver in attempting to
%               construct d1 . A more robust QP solver may succeed.
%            info = 7: Input data are not consistent (with printout
%               indicating the error).
%            info = 8: Two consecutive iterates are numerically
%               equivalent before a stopping criterion is satisfied.
%            info = 9: One of the penalty parameters exceeded
%                BIGBND. The algorithm is having trouble satisfying a
%                nonlinear equality constraint.
%
%
%     METHOD
%
%     If the initial guess provided by the user is infeasible for
%     nonlinear inequality constraints and linear constraints, FFSQP
%     first generates a point satisfying all these constraints by
%     iterating on the problem of minimizing the maximum of these
%     constraints.
%
%     Then, using Mayne-Polak's scheme, nonlinear equality
%     constraints are turned into nonlinear inequality constraints and the
%     original objective function is replaced by a modified objective
%     function.
%
%     After obtaining feasibility, either (i) an Armijo-type line
%     search may be used (algorithm FFSQP-AL), yielding a monotone
%     decrease of the objective function at each iteration; or (ii) a
%     nonmonotone line search may be selected (algorithm FFSQP-NL),
%     forcing a decrease of the objective function within at most four
%     iterations.
%
%     See [1] for further details.
%
%     REFERENCES
%
%     [1] J. L. Zhou, A. L. Tits and C. T. Lawrence
%         User's Guide for FFSQP Version 3.7 : A Fortran Code for
%         Solving Optimization Programs, Possibly Minimax, with General
%         Inequality Constraints and Linear Equality Constraints,
%         Generating Feasible Iterates
%         Institute for Systems Research, University of Maryland,
%         Technical Report SRC-TR-92-107r5, 1997.
%         (Available at http://gachinese.com/aemdesign/FSQPframe.htm)
%
%     NUMERICAL ASPECTS
%
%     CONTRIBUTOR
%
%     Fernando Alvarruiz, Vicente Hernandez
%     Universidad Politecnica de Valencia, Spain
%
%     REVISIONS
```

```
%
%     KEYWORDS
%
%     Sequential Quadratic Programming, Non-Linear Programming
```